KIRGIZİSTAN TÜRKİYE MANAS ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI

BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA

YÜKSEK LİSANS TEZİ

Mehmet Selim ELMALI

BİŞKEK 2010

KIRGIZİSTAN TÜRKİYE MANAS ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI

BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA

YÜKSEK LİSANS TEZİ

Mehmet Selim ELMALI

Danışman
Doç. Dr. Rayımbek SULTANOV

BİŞKEK 2010

TUTANAK

Kırgızistan-Türkiye Manas Üniversitesi Fen Bilimler Enstitüsü'nün ***01/06/2010*** tarih ve ***4 sayılı*** toplantısında oluşturulan jüri, Lisansüstü Öğretim ve Sınav Yönetmeliğinin (yüksek lisans için 36) maddesine göre Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans öğrencisi ***0751Y01004*** numaralı ***Mehmet Selim Elmalı "Bilgisayar Mimarisi Öğretimi için Basit 8-Bitli İşlemci Emülasyonu Üzerine Bir Çalışma"*** konulu tezini incelemiş ve aday 18/06/2010 tarihinde, saat 11:00 de jüri önünde tez savunmasına alınmıştır.

Adayın kişisel çalışmaya dayanan tezini savunmasından sonra 35 dakikalık süre içinde gerek tez konusu ve tezi gerekse tezin dayanağı olan anabilim dallarından jüri üyelerinin sorularına verdiği cevaplar değerlendirilerek tezin **Kabulüne** oy birliği/çokluğu ile karar verildi.

Başkan:	Prof. Dr. Cumabay Usenkanov	(İmza)
Üye:	Prof. Dr. Ulan Brımkulov	(İmza)
Üye:	Doç. Dr. Rayımbek Sultanov	(İmza)
Üye:	Dr. Bakyt Şarşembaev	(İmza)
Üye:	Dr. Şuhrat Nasirov	(İmza)

ЧЕЧИМ

Кыргыз-Түрк Манас университетинин Табигый Илимдер Институтунун экзамендик инструкциясынын 36-жобосуна ылайык, ***2010 жылындагы 1 июндагы 4 жыйында*** уюшулган комиссия, Компьютердик Инженерия Бөлүмүнүн магистранты ***Мехмет Селим Елмали "Компьютердин архитектурасын үйрөтүү үчүн жөнөкөй 8-биттик процессорун эмуляциясы жөнүндө бир иш"*** темасында жазган дипломдук проектин анализдеп, **18/06/2010-ж. Саат** 11 : 00 жактоого кабыл алды.

Магистрант 35 минута убакыт ичинде дипломдук проектин жактап, комиссия *көпчүлүк добуш менен/бир добуштан* ***Кабыл алынсын*** деген чечим чыгарды.

Жюри төрагасы: ф-м.и.д., доц. Усенканов Жумабай Осмонбекович

Жюри мүчөсү: т.и.д., проф. Бримкулов Улан Нургазиевич

Жюри мүчөсү: ф-м.и.к., доц. Султанов Райымбек Касымович

Жюри мүчөсү: ф.и.к., Шаршембаев Бакыт Догдурович

Жюри мүчөсү: ф.-м.и.к. Насиров Шухрат

İntihal Yapılmadığını Belirten İfade

Ben bu tezdeki bütün bilgileri akademik ve etik kurallarına göre aldığımı ve sunduğumu belirtiyorum. Bu çalışmaya özgün olmadan kullandığım bütün materyal ve bilgilere akademik ve etik kurallar gereğince atıfta bulunduğumu ve hiçbir şekilde intihal yapmadığımı belirtiyorum.

AD, SOYAD: **Mehmet Selim ELMALI**
İMZA :
TARİH : 18.06.2010

Плагиат жасалбагандыгы тууралуу билдирүү

Мен бул эмгекте алынган бардык маалыматтарды академиялык жана этикалык эрежелерге ылайык колдондум. Тагыраак айтканда бул эмгекте колдонулган бирок мага тиешелүү болбогон маалыматтардын бардыгын тиркемеде так көрсөттүм жана эч кайсы жерден плагиат жасалбагандыгына ынандырып кетким келет.

АТЫ, ЖӨНҮ : **Mehmet Selim ELMALI**
КОЛУ :
ДАТАСЫ : **18.06.2010**

ÖZ

Yazar	: Mehmet Selim Elmalı
Üniversite	: Kırgızistan -Türkiye Manas Üniversitesi
Anabilim Dalı	: Bilgisayar mühendisliği
Bilim Dalı	:
Tezin niteliği	: Yüksek Lisans Tezi
Sayfa Sayısı	: 82
Mezuniyet Tarihi	: 29.05.2010
Tez danışmanı	: Doç. Dr. Rayımbek SULTANOV

BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA

Bilgisayar mimarisi derslerini öğrenirken gerçek bir mikroişlemcinin mimarisini, o işlemcinin makine dili komutlarını ve o işlemciyi çevirme dilinde(assembly language) programlayarak öğrenmek önemlidir.

Öğrenciler, makine dili modellemesi ile, bir donanımı kurup onun emülasyonunu gerçekleştirdikleri zaman daha iyi öğrenirler. Multimedia Logic (MML) grafiksel bilgisayar mimarisi, uygun G/Ç desteği sağlayarak bilgisayar tasarımı ve emülasyonuna olanak sağlayan yazılımlardan biridir. Ancak MML üzerine kurulan emülatörler, MML ye bağımlı kalmaktadır.

Bu çalışmada 8-bitli basit bir mikro işlemci emülatörü ve bu emülatörün assembly dili tasarımı yapılmıştır. MML den bağımsız olması için de emülatör C++ dilinde kodlanmıştır. Çalışmada yaygın olarak kullanılmış başarılı bir ürün olması, basitliği ve kolay anlaşılır olması nedeniyle Motorola 6800 işlemcisini komutları örnek alınmıştır.

En sık kullanılan makine dili komutları, akümülatör yazmacına değer yükleyen ve akümülatördeki değerleri işleyen (toplama, belleğe saklama gibi) komutlardır. Bu tezde tasarlanan işlemcinin bir adet 8-bit akümülatörü (“A”), bir adet 16 bit indeks yazmacı (IX), ve 16-bit program sayacı (PC) vardır.

Akümülatör komutları:

LDA <değer> – “A” akümülatörüne bir değer yükle
STA <bellek adresi> - “A” akümülatörünün değerini belleğe kopyala
ADA <değer> veya <bellek adresi>

Diğer önemli komut kümesi program akışını dallandırma (“branch”) komutlarıdır. Bu komutlarda “göreceli” adresleme kullanmakta olup “-128” ile “+127” arasında bir değeri program sayacı (PC) yazmacına ekleyerek program kontrolünü elde edilen bu yeni adresteki komutu bir sonraki komut olarak çalıştıracak şekilde değiştirmek mümkündür. Böylece değişik koşullar altında programın değişik kısımlarının çalıştırılması ve farklı işlem yapılması gerçekleştirilmektedir.

Bu komutların yeterli işlevselliği sağlayacak küçük bir alt kümesi bu çalışmada gerçekleştirilen emülatör tarafından tanınmakta ve emüle edilmektedir, Bu komutlar aşağıda verilmiştir.

Dallanma komutları:

BRZ	Eğer “sıfır” ise
BEQ	Eğer “eşit” ise
BLT	Eğer “küçüktür” sıfır ise
BGT	Eğer “büyüktür” sıfır ise
BNE	Eğer “eşit değil” ise

Aritmetik ve mantık kaydırma komutları akümülatörün içindeki değeri sola veya kaydırma ve döndürme işlemlerinin gerçekleştirmek için kullanılırlar. Bu çalışmadaki emülatör aşağıdaki kaydırma ve döndürme komutlarını gerçekleştirebilmektedir.

Kaydırma ve döndürme komutları:

ASL	Aritmetik sola kaydır
ASR	Aritmetik sağa kaydır
LSR	Mantıksal sağa kaydır
ROR	Sağa döndür
ROL	Sola döndür

Diğer bir önemli komut kümesi de STATUS (durum) yazmacında bulunan C (carry/elde), Z (zero/sıfır), N (negatif) gibi bayrak değerlerini değiştiren komutlardır.

SEC C = 1
CLC C = 0
SEZ Z = 1
CLZ Z = 0
SEN N = 1
CLN N = 0

İşleyiş ve veri tanımlama komutları:

“ORG” : programın bellekte başlangıç adresini vermektedir.
“DB” DEFINE BYTE: Bellekte önceden tanımlanmış değerlerin yerleştirilmesine izin vermektedir.
“END” : Programın çalışmasını sona erdirmektedir.

Emülatörü test etmek için bazı deneme uygulamaları yapılmış ve LDA, STA, ADD, BRA, BNE, ASL, ASR gibi temel komutların doğru şekilde çalıştığı görülmüştür.

Geliştirilen emülatör C++ dilinde kodlanmıştır. Bu sayede temel foksiyonlar C++ dilinde kodlandığı için yeni bir mikro işlemci mimarisi ve/veya makine dili tasarlanarak kolay bir şekilde emülatörün içine eklenebilecektir.

Anahtar sözcükler: bilgisayar, mimarisi, yapısı, mikroişlemci, simülatör, simülasyon, emülatör, emülasyon.

КЫСКАЧА МАЗМУНУ

Дайардаган	: Мехмет Селим Элмалы
Университет	: Кыргыз-Түрк “Манас” Университети
Институт	: Табигий Илимдер Институту
Багыты	: Компьютер инженериясы
Иштиң сыпаты	: Магистратура
Беттердин саны	: 83
Бүтүүүү датасы	: 29 Апрел 2010
Диссертация жетекчиси	: Dr. Rayımbek SULTANOV

КОМПЬЮТЕРДИН АРХИТЕКТУРАСЫН ҮЙРӨТҮҮ ҮЧҮН ЖӨНӨКӨЙ 8-БИТТИК ПРОЦЕССОРДУН ЭМУЛЯЦИЯСЫ ЖӨНҮНДӨ БИР ИШ

“Компьютердин архитектурасын” окуу курсунда реалдык микропроцессорду окуп түшүнүү үчүн – микропроцессордун түзүлүшүн, машиналык командаларын – микропроцессордун ассемблер тилин окуп үйрөнүү өтө маанилүү.

Студенттер бул айтылган сабакты окуп үйрөнүүдө процессорду курганды жана аны техникалык камсыздыгын эмуляция кылганды, машина тилин моделдештирип имитация кылып үйрөнсө сабакты жакшы өздөштүрүшөт. Мисалы бул Multimedia Logic (MML) болушу мүмкүн, ал кандайдыр бир графикада компьютердин архитектурасы болот жана анда кириш-чыгыш мүмкүнчүлүктөрү бар. Бирок MML колдонуп курулган эмуляторлор, MMLден көз каранды болот.

Бул жумушта 8 биттик микропроцессорду эмуляциялоо программасы каралат, анда микропроцессордун түзүлүшү жана аны менен байланышкан ассемблер тилинин иштөөсү каралат. MMLден көз каранды болбош үчүн эмуляциялоо программасы C++ тилинде жасалган. Motorola 6800 микропроцессорунун архитектурасы жана ага туура келген машина тили мисал катары көрсөтүлгөн. Ал жөнөкойлүгү менен окуп үйрөнүүгө жакшы.

Машина тилинде өтө көп колдонуучу командалардын бири болуп кандайдыр бир маанини машинага жүктөо жана аны менен болгон амалдар-командалар эсептелет. Биз түзгөн микропроцессор 8 биттик аккумулятору(A) болот, анын 16 биттик индекстөөчү регистри бар (IX) жана 16 биттик командаларды эсептөөчү регистрден турат (PC).

Accumulator командалары

LDA <маани>: “A” аккумуляторунун регистрине маанини жүктөө. Бул командanын параметринин мааниси кезектеги аткарылуучу командanын номери же эстин адреси болушу мүмкүн(түз, кеңейтилген же индекстүү түрдө).
STA <эстин адреси>: “A” аккумуляторуна эстин адресин жүктөйт. Адрес төмөнкү түрлөрдүн бири болушу мүмкүн - direct mode (1 байт, башкача айтканда 0..255 чейинки бүтун сан), extended mode же indexed mode.
ADA < маани >: “A” аккумуляторуна маанини кошот.

Тармактануу командалары

Негизги командалардын көптүгүнө тармактануу командалары кирет жана алар программанын аткарылуу ирээтин өзгөртүп туруу үчүн колдонулат. Тармактануу командаларында “салыштырмалуу адресс” колдонулат. Алардын мааниси -128 тен +127 ке чейин болушу мүмкүн. Ал регмстрге жүктөлөт жана кийинки аткарылуучу командanын адресин көрсөтөт.

Алардын кээ бир түрлөрү бул эмулятордун функционалдуулугуна зарыл болгондору бул иште колдонулган. Алардын түрлөру:

BRZ	тармактан эгерде ноль
BEQ	тармактан эгерде барабар
BLT	тармактан эгерде кичине
BGT	тармактан эгерде чоң
BNE	тармактан эгерде барабар эмес

Арифметикалык жана логикалык командалар регистрдин маанисин солго же оңго жылдыруу жана өзгөртүү үчүн колдонулат.

Жылдыруу жана ротация командалары

ASL	солго арифметикалык жылдыруу
ASR	оңго арифметикалык жылдыруу
LSR	оңго логикалык жылдыруу
ROR	оңго ротация
ROL	солго ротация

Регистрдин желекчесин STATUSун өзгөртүү үчун төмөку командалар көптүгү колдонулат: C(өзгөртүү), Z(нөл) и N(оң эмес) желекчклери.

SEC C = 1
CLC C = 0
SEN N = 1
CLN N = 0
SEZ Z = 1
CLZ Z = 0

Программаны башкаруу командалары жана программада колдонулуучу маанилерди аныктоочу командалар дагы бул микропроцессорду түзүүдө колдонулган, ошондой эле алардын кодун машина тилине генерацялап эске сактоо каралган:

ORG(origin): бул команда аркылуу ассемблер колдонуучу эстин областы көрсөтулөт.

DB(АНЫКТАМА BYTE): прогамманы башталган чекитин көрсөтүү учүн колдонулат.

END: ассемблер программасынын аягын көрсөтөт.

Эмулятору текшерүү (тестирлөө) үчүн жасалган жана бул проекте колдонулган ассемблер тилиндеги төмөнкү командалар кирет: LDA, STA, ADD, BRA, BNE, ASL жана ASR.

Эмулятор C++ тилинде жазалган. Бул C++ тилин колдонуу эмуляторду өзгөртүүгө же жаңы микропроцессорлорду проектирлөөдө жана ошондой эле жаңы машина тилиндеги командаларды кошуп толуктоодо абдан ыңгайлуу.

Ачкыч сөздөр: *bilgisayar, mimarisi, structure, microprocessor, simulator, simulation, emulator, emulation.*

ABSTRACT

Author	: Mehmet Selim Elmalı
University	: Kyrgyzistan -Türkiye Manas University
Main Discipline	: Natural Sciences
Department Name	: Computer Engineering
Thesis Type	: Masters Thesis
Number of pages	: 83
Graduation Date	: 29.05.2010
Thesis Supervisor	: Assoc. Prof. Dr. Rayımbek SULTANOV

A STUDY ON SIMPLE 8-BIT PROCESSOR EMULATION FOR TEACHING COMPUTER ARCHITECTURE

In teaching computer architecture courses it is important to learn a real microprocessor, its architecture and the associated machine language instructions and assembly language programming using that microprocessor.

Students learn better when they actually build (without adding the complexity of hardware details) and emulate hardware using machine language modeling and emulation. An example of this is the Multimedia Logic (MML) which provides a graphical computer architecture, convenient I/O support and enables building and emulating computers. But emulator designs built on MML, become dependant on MML.

In this thesis a simple 8-bit microprocessor emulating software and associated assembly language has been designed. And to make it independent from MLL, it is coded in C++. The Motorola 6800 microprocessor architecture and machine language commands have been taken as examples because of their simplicity and easily understandable nature.

The most frequently used machine language commands are those that load values and manipulate (e.g., adding, storing to memory) values in working (accumulator) registers. The microprocessor designed in this thesis has one 8-bit accumulator (A), one 16-bit indexing register (IX), and one 16-bit program counter (PC) register.

Accumulator commands

LDA <value>: load a value to the accumulator register, A. The value can be an "immediate" parameter (in the next byte in memory following the machine language command), or a memory address (direct, extended, or indexed mode) given as parameter to the command.
STA <memory address>: Save the value of "A" accumulator at the given memory address. The address can be either direct mode (1 byte in the memory addresses 0..255), extended mode, or indexed mode.
ADA <value>: Add the given value to the contents of accumulator "A".

Branching commands

The second set of important commands are branching commands used for modifying the execution sequence (normally in increasing order of memory addresses) of program commands. The branching commands use "relative addressing" where a value between -128 and +127 is added to the current PC register value to obtain the memory address of the next machine language command and store this value in PC register to point at the next command to be executed. This ability to change program flow allows different parts of the code to be executed under different conditions and achieve different operations to be accomplished.

A smaller subset of these commands, sufficient to establish functionality, has been implemented in this work and are recognized and executed by the developed emulator program. These commands are given below:

BRZ	branch if zero
BEQ	branch if equal
BLT	branch if less than
BGT	branch if greater than
BNE	branch if not equal to

Arithmetic and logical shitfting commands are used for shifting or rotating the value in accumulator register left or right. The emulator developed in this thesis recognizes and can execute the following shift and rotate commands.

Shifting and rotation commands

ASL	arithmetic shift left
ASR	arithmetic shift right
LSR	logical shift right
ROR	rotate right
ROL	rotate right

Another important set of commands modify the flags in the STATUS register, some of which are; C(carry), Z(zero), and N(negative) flags.

SEC C = 1
CLC C = 0
SEN N = 1
SEZ Z = 1
CLZ Z = 0
CLN N = 0

Program control and data definition commands are also provided for controlling the placement of program data and generated machine language code in memory:

ORG(origin): this command sets the current memory address to be used by the assembler.

DB(DEFINE BYTE): This command allows a predefined value to be inserted into the memory by the assembler, before the program execution starts.

END: This program terminates the operation of the assempler.

In order to test the emulator several small assembly language application code have been written using LDA, STA, ADD, BRA, BNE, ASL, and ASR commands and they have been verified to be correctly emulated by the developed program.

The developed emulator is written in C++ language. The use of C++ also makes it possible to add (relatively easily) newly designed microprocesor architectures and/or new machine languages into the program developed in this thesis.

Keywords: *bilgisayar, mimarisi, structure, microprocessor, simulator, simulation, emulator, emulation.*

АБСТРАКТ

Автор	: Mehmet Selim Elmalı
Университет	: Кыргызско-Турецкий Университет “Манас”
Институт	: Инстит Естественних Наук
Кафедра	: Компьютерная инженерия
Качество диссертации	: Магистратура
Количество страниц	: 83
Дата выпуска	: 29 Апрел 2010
Руководитель диссертации	: Доцент Док. Райымбек Султанов

ПРИМЕНЕНИЕ ЭМУЛЯЦИИ ПРОСТОГО 8-БИТНОГО ПРОЦЕССОРА ДЛЯ ИЗУЧЕНИЯ АРХИТЕКТУРЫ КОМПЬЮТЕРА

В изучении курса “Архитектура компьютера” очень важным является изучение реального микропроцессора, его архитектуры и команд машинного языка – языка ассемблер, используемого данным микропроцессором.

Студенты лучше обучаются данному предмету, когда они фактически строят (без учета сложности деталей архитектуры) и эмулируют техническое обеспечение, используя моделирование и имитацию машинного языка. К примеру, это может быть the Multimedia Logic (MML), который обеспечивает некоторую графическую компьютерную архитектуру, удобную поддержку устройств ввода-вывода и делает возможным эмуляцию сборки компьютера. Но эмулятор который использует MML будет зависимый от MML.

В данной работе рассматривается программа которая эмулирует 8-битный микропроцессор, ассоциурующий ассемблерный язык с его конструкцией. Чтобы быть независимый от MML используется для постороение эмулятора язык C++. Архитектура микропроцессора Motorola 6800 и команды машинного языка даны как пример, так как они просты и легко в понимании природы работы устройства.

Часто используемыми командами машинного языка являются те команды, которые загружают значения и манипулируют ими (например добавить и сохранить в память) в рабочих регистрах. Сконструктированный в данной работе микропроцессор имеет 8-битный аккумулятор (A), один 16-битный индексируемый регистр (IX) и 16-битный регистр счетчика команд (PC).

Команды accumulator

LDA <значение>: загрузка значений в регистр аккумулятора “A”. Значение можеть быть текущим параметром команды (в следующем за командой машинного языка байте памяти) или адресом памяти (в прямом, расширенном или индексированном режиме).
STA <адрес памяти>: сохраняет значение “A”-аккумулятора в указанный адрес памяти. Адрес может быть одного из следующих типов - direct mode (1 байт, то есть один из адресов 0..255), extended mode или indexed mode.
ADA < значение >: добавляет данное значение в содержимое аккумулятора “A”.

Команды ветвления

Следующим подмножеством основных команд является команды ветвления, используемые для модификации выполнения последовательности команд выполнения программы. В командах ветвления используются «относительные адреса», значения которых могут быть от -128 до +127, они добавляются к текущему значению регистра и дают следующий адрес памяти для выполнения.

Некоторые подмножества этих команд, необходимых для установки функциональности, были реализованы в данной работе, распознются и выполняются разработанным эмулятором программ. Это следующие команды:

BRZ	ветвление если ноль
BEQ	ветвление если равно
BLT	ветвление если менее чем
BGT	ветвление если больше чем
BNE	ветвление если не равно

Арифметические и логические команды, используемые для сдвига и перемещения значений регистра налево и направо..

Команды сдвига и перемещения
Shifting and rotation commands

ASL	арифметический сдвиг налево
ASR	арифметический сдвиг направо
LSR	логический сдвиг направо
ROR	перемещение напрво
ROL	перемещение налево

Другие важные множество команд для модификации STATUS флага регистра, такие как; C(переносить), Z(ноль) и N(отрицательный) флаги.

SEC C = 1
CLC C = 0
SEN N = 1
CLN N = 0
SEZ Z = 1
CLZ Z = 0

Команды управления программой и определения данных также поддерживаются для контроля размещения данных программ и генерации кодов машинного языка в памяти:

ORG(origin): данная команда устанавливает текущую область памяти для использования ассемблером.

DB(ОПРЕДЕЛЕНИЯ BYTE): команда, которая позволяет предопределить значение, которое должно быть вставлено в память о начала выполнения программы.

END: конец выполнения программы ассемблера.

Для проверки (тестирования) эмулятора были написаны и использованы приложения на языке ассемблер с использованием команд LDA, STA, ADD, BRA, BNE, ASL и ASR.

Эмулятор разработан на языке C++. Использование C++ также дает возможность добавления в разработанную в рамках данного проекта программу новых типов микропроцессора и/или нового машинного языка.

Ключевые слова: *bilgisayar, mimarisi, structure, microprocessor, simulator, simulation, emulator, emulation.*

ÖNSÖZ

Bu çalışma, bilgisayar mimarisi ve bilgisayar yapısı derslerinde, öğrencilerin, bir mikroişlemcinin bölümlerini yapısını ve çalışmasını kafalarında daha iyi canlandırabilmek, assembly dili kavramını ve komutlarını daha iyi anlayabilmelerini sağlamak amacıyla yapılmıştır.

Çalışmada anlaşılması kolay olması bakımından basit 8-bitli bir işlemci emülatörü tasarlanmıştır.

Emülatörün tasarımı bittikten sonra bazı örnek uygulamalarla testi yapılmıştır. Testte kullanılan örnek uygulamalar ve test sonuçları çalışmada verilmiştir.

Bu çalışma sırasında değerli yardımlarını esirgemeyen,

Doç. Dr. Sayın Raimbek Sultanov(Kırgızistan-Türkiye Manas Üniversitesi, Bilgisayar Mühendisliği Bölümü)

ve

Doç. Dr. Sayın Ahmet Coşar'a(O.D.T.Ü. Bilgisayar Mühendisliği Bölümü) teşekkürü bir borç bilirim.

İÇİNDEKİLER

KISALTMALAR

Kısaltma	Bibliografik bilgi
ADC	Add with Cary
AMB	Aritmetik Mantık Birimi
AS	Address Select
C	Carry
CS	Chip Select
Çe	Çıkan elde
DAA	Decimal Adjust Accumulator
DB	Define Byte
G/Ç	Giriş/çıkış
Ge	Giren elde
MİB	Merkezi İşlem Birimi
MML	Multimedia Logic
N	Negative
PC	Program Counter
R/W	Read/Write
ROM	Read Only Memory
SP	Stack Pointer
TD	Tümleşik Devre
VMA	Valid Memory Address
Z	Zero

ÇİZELGELER LİSTESİ

ŞEKİLLER LİSTESİ

SİMGELER

Simge	Anlamı
*	Çarpma
<	Küçük
>	Büyük
=	Eşit
MEM[100-119]:	100 den 119 a kadar olan bellek adresinin içeriği

GİRİŞ

Öğrenciler, bilgisayar mimarisini ve yapısını öğrenirken gerçek bir mikroişlemci tasarımı yaparak bunu donanım olarak gerçekleştirdikleri zaman veya bunu yerine onu tasarlayıp emülasyonunu gerçekleştirdikleri zaman daha iyi öğrenir ve konuyu daha iyi kavrarlar.

Bilgisayar Mimarisi ve Bilgisayar Yapısı derslerinde bilgisayarın çalışmasının ince ayrıntılarını, günümüzün ileri bilgisayar teknoloji düzeyinde, gerçek donanım üzerinde uygulama yaparak gerçekleştirmenin birtakım zorlukları bulunmaktadır [1]. Bu nedenle, bilgisayar mimarisi öğretimi için gerçek donanım kullanılarak işlemci tasarımı yapılabildiği gibi [2], bu amaçla işlemci simülatörleri de kullanılabilmektedir. Bazı uygulamalarda ise simülatörler ve gerçek donanım birlikte kullanılabilmektedir [3].

Bilgisayarın çalışmasını simüle etmek amacıyla geliştirilmiş yazılımlarda birisi de MML(Multi Media Logic) simülatörüdür. MML, bilgisayar donanımının simülasyonunda ve bilgisayar mimarisinin ve yapısının öğretiminde yardımcı olarak, en çok tercih edilen yazılımlardan biridir. . MML yazılımı grafiksel bir bilgisayar mimarisi gösterimi, kolay Girdi/Çıktı desteği ile bilgisayar inşasına ve emülasyonuna olanak vermektedir. Bu simülatörle bilgisayarın üzerindeki gerçek aygıtlara doğrudan bağlanabilen kavramsal aygıtlar (tuş takımı, ekran, seri kapılar (port) gibi) ve ayrıca çoklu ortam (PC hoparlörü, ses aygıtı, ve ikonlar gibi) aygıtları simüle edilebilmektedir. Bu yöntemler yardımıyla öğrenciler donanımın iç işleyişini ve ayrıntılarını çok daha iyi ve kısa sürede anlayabilmektedirler.

2005 ve 2007 yıllarında MML kullanılarak başarılı bazı çalışmalar yapılmıştır [4] [5]. MML simülasyonda oldukça başarılı bir yazılımdır ancak bunun üzerine kurulan çalışmalar da MML ye bağımlı olmaktadırlar. 2006 yılında MML kullanılmadan değişik mimarileri içeren bir çalışma yapılmıştır, ancak bu çalışmada işlemcinin çalışması sırasında yazmaçların ve belleğin durumu görülememektedir [5].

Bilgisayar mimarisi derslerinde gerçek bir mikroişlemcinin mimarisinin, makine dilinin ve assembler dili programlamasının öğrenilmesi gerekir. Makine dilinin doğrudan emülasyonunu yapan bir yazılım kullanarak, daha iyi bir öğrenme sağlanacak ve eğitimde iyileşme sağlanmış olacaktır.

Bu tezde, üniversitelerde verilen bilgisayar mimarisi derslerinde, öğrencilerin MİB nin bölümlerinin yapısını ve çalışmasını daha iyi anlamalarını sağlayacak, örnek bir mikroişlemci tasarımı yapılması, bir assembler tarafından üretilen makine dili kodlarının bir emülatör aracılığı ile çalıştırılması ve makine düzeyi komutların yazmaçlar üzerindeki etkisinin doğrudan öğrenciler tarafından görülebilmesi amaçlanmaktadır.

Bu amaçla, tezde, MİB(Merkezi İşlem Birimi) nin çalışmasını emüle eden bir emülatör ve assembly dili geliştirilmiştir. Emülatörün, MML ye bağımlı olmaması için yazılım, C++ dilinde kodlanmıştır.

İşlemci mimarisine örnek olarak bu tezde, Motorola 6800 mikro işlemcisi hem yaygın olarak kullanılmış başarılı bir ticari ürün olması, hem de eğitim öğretim bakımından kolay materyal ve yazılım bulunan, literatürde dokümanları gayet iyi hazırlanmış, tasarım ve çalışma ayrıntıları çok iyi bilinen bir mikro işlemci olması nedenleriyle tercih edilmiş ve temel alınmıştır [7].

Bir makine simülatörü, bellek ve yazmaçların içeriğini, içerdiği assembler aracılığı ile de bellekteki makine dili komutlarını gösterebilir. Bu komutları birer birer çalıştırarak ve komutların bellekteki etkileri gözlenerek donanımın çalışması ve makine dilinde yazılım geliştirme daha iyi anlaşılabilir.

Bu tezin Birinci Bölümünde genel mikroişlemci mimarisi Motorola 6800 mikroişlemci sinin mimarisi, emülatörler ve makine dili konusunda gerekli bilgiler verilmektedir,.

İkinci Bölüm de, örnek bir 8-bit işlemci makine dili tasarlanmakta, ayrıntıları

sunulmakta ve bu makine dilinin komutlarını emüle eden bir C++ yazılımının yapısı açıklanmaktadır. Üçüncü Bölüm de, geliştirilen bu makine dili ve emülatör ile yapılmış olan örnek assembler kod denemeleri yapılmakta ve bu kodların çalışması ve sonuçları açıklanmaktadır. Sonuç bölümde ise geliştirilen bu makine dili ve emülatör'den edinilen deneyim anlatılmakta ve eğitim amaçlı olarak bu makine dili ve emülatör yazılımının nasıl kullanılacağı ve daha da geliştirilmesi için nelerin yapılabileceği tartışılmaktadır.

1. BÖLÜM

GENEL BİLGİLER

1.1. Bilgisayar Mimarisi

İşlemci mimarisine örnek olarak bu tezde Motorola 6800 mikro işlemcisi[Motorola] hem yaygın olarak kullanılmış başarılı bir ticari ürün olması, hem de eğitim öğretim bakımından kolay materyal ve yazılım bulunan, literatürde dokümanları gayet iyi hazırlanmış, tasarım ve çalışma ayrıntıları çok iyi bilinen bir mikro işlemci olması nedenleriyle tercih edilmiştir. Bu işlemcinin komutları hakkında ayrıntılı bilgi Çizelge 1.1, Çizelge 1.2 ve Çizelge 1.3 de verilmektedir.

Çizelge 1.1 6800 işlemcisinin komut örnekleri.

Komut	Tanımı
NOP	Işlem yok
TPA	Status flag yazmacını A'ya transfer et
INX	Indeks yazmacını 1 artır
DEX	Index yazmacını 1 azalt
SBA	B yazmacını A'dan çıkar
CBA	B yazmacını A ile karşılaştır

Çizelge 1.2 6800 işlemcisinin adrese git komutları

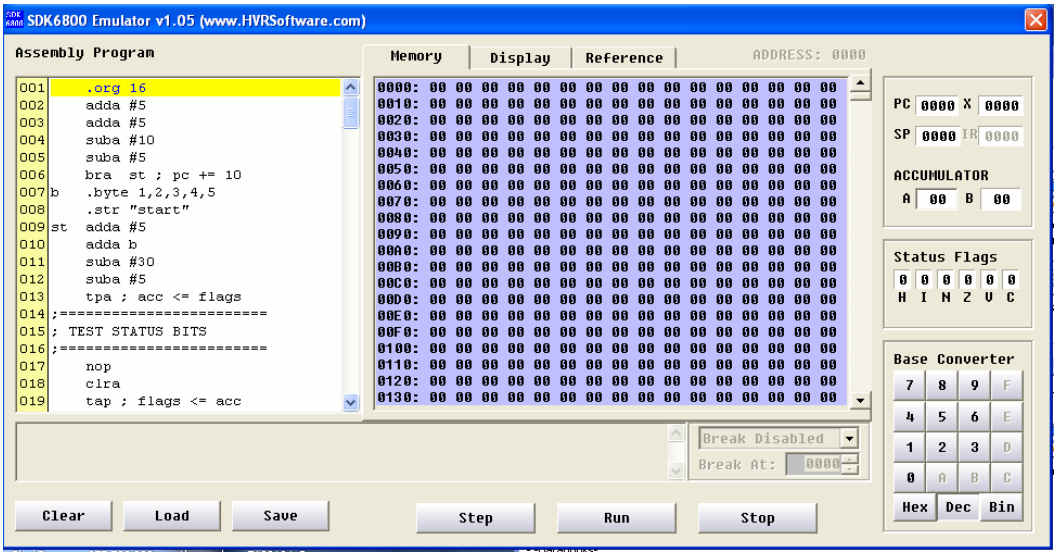
Komut	Açıklama
BRA	Koşulsuz olarak adrese git
BHI	ACCUM daha büyükse adrese git
BLS	ACCUM daha küçük veya aynıysa
BCC	C=0 ise git
BCS	C=1 ise git
BNE	Z=0 ise git
BEQ	Z=1 ise git
BVC / BVS	V=0 / V=1 ise git
BGE	A >= ise git
BLT	A < ise git
BGT	A > ise git
BLE	A <= ise git

Çizelge 1.3. 6800 işlemcisinin yığıt işleme komutları.

Komut	Açıklama
TSX	Yığıt göstergecini Indeks yazmacına aktar
INS	Yığıt göstergecini artır
PUL A	Yığıt'tan A'ya veri aktar
PUL B	Yığıt'tan B'ye veri aktar
DES	Yığıt göstergecini 1 azalt
TXS	Indeks yazmacını yığıt göstergecine aktar
PSH A	A yazmacının içeriğini yığıtın üstüne koy
PSH B	B yazmacının içeriğini yığıtın üstüne koy
RTS	Alt yordamdan geriye dön
JTS	Alt yordama git

1.2. Bilgisayar/Mantık Simülatörleri

Bir makine simülatörü bellek ve yazmaçların içeriğini, içerdiği assembler aracılığı ile de bellekteki makine dili komutlarını gösterebilir. Bu komutları birer birer çalıştırarak ve komutların bellekteki etkilerini gözleyerek donanımın çalışmasını ve makine dilinde yazılım geliştirmeyi daha iyi anlayabiliriz. Şekil 1.1. de örnek bir emülatör yazılım ve ekran görüntüsü görülmektedir.

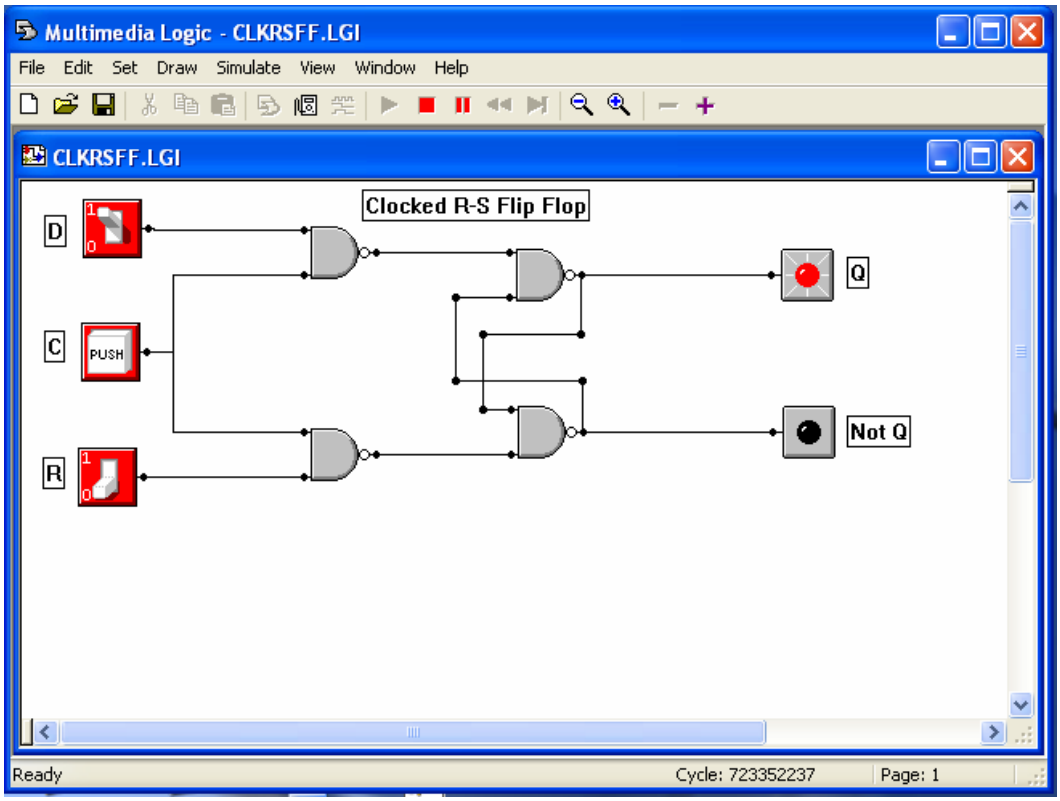


Şekil 1.1. Örnek bir emülatör yazılımı

1.2.1. Multimedia Logic (MML) Simülatörü

MML simülatörü bilgisayarın üzerindeki gerçek aygıtlara doğrudan bağlanabilen kavramsal aygıtları (tuş takımı, ekran, seri kapılar (port) gibi) ve ayrıca çoklu ortam aygıtlarını içermektedir (PC hoparlörü, ses aygıtı, ve ikonlar gibi)[8]. Şekil 1.2 de basit bir RS-flip flop ve bağlı aygıtlar görülmektedir. Burada görülen "D" girdisi flip-flop'un "1", "R" düğmesi ise "0" konumuna geçmesine komut vermektedir. Ancak bu komutun uygulanabilmesi için "C" düğmesi ile "clock" sinyalinin el ile üretilmesi gerekir. Bu örnek üzerinde öğrenciler bir RS flip-flop'un çeşitli denetim sinyallarıyla nasıl "0/1"

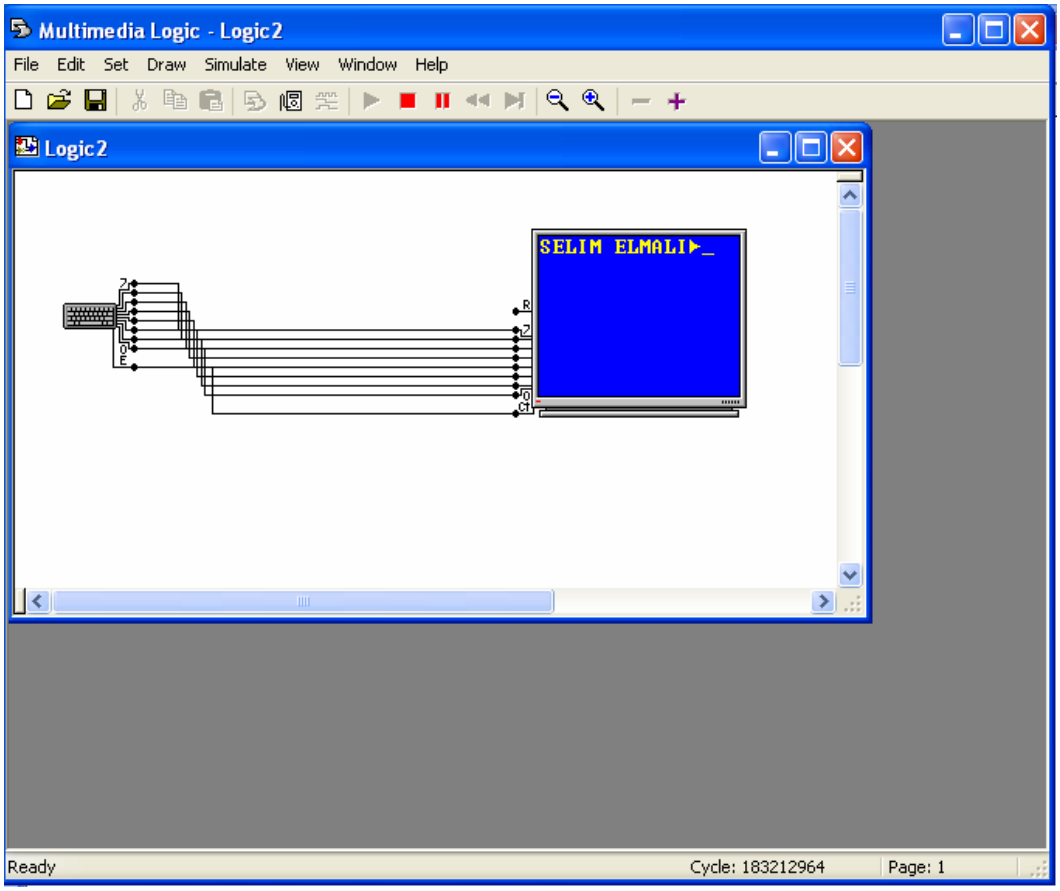
durumlarına girdiğini görerek öğrenimlerini pekiştirebileceklerdir.



Şekil 1.2. Bir R-S tipi flip-flop emülasyonu(MML).

Şekil 1.3 te ise MML tarafından sağlanan sanal girdi ve çıktı cihazları olan tuş takımı ve ekran birbirlerine bağlanmıştır. Tuş takımından bir tuşa basıldığında tuş takımının "E" (enable) çıkışı "1" durumuna geçer ve bu çıkışı ekran cihazının "C" (clock) giriş hattına bağlayarak tuş takımının çıkışlarında geçerli sinyallar olduğunda bu sinyallar ekran cihazının giriş hatlarına kopyalanmaktadır.

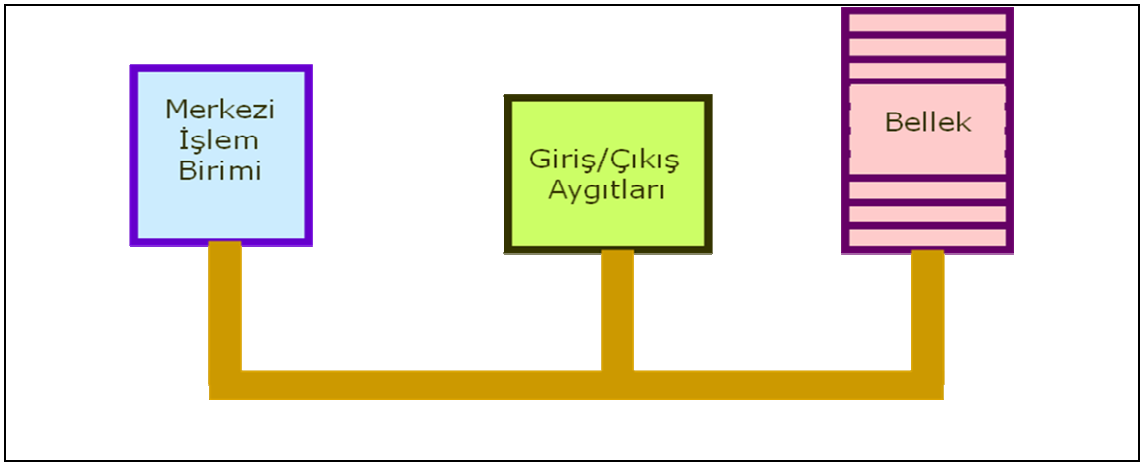
.



Şekil 1.3. Tuş takımı ve ekran bağlantısı.

1.3. Bilgisayar Mimarisi

Günümüzde kullanılan bilgisayarların büyük çoğunluğu, öbek çizimi Şekil 1.4. de gösterilen Von Neumann mimarisine sahiptir.



Şekil 1.4. Von Neumann mimarisi.

1.3.1. Yazmaçlar

Yazmaçlar son derece hızlı bellek hücrelerinden oluşur. AMB lerdeki yazmaç sayıları 32 ye kadar çıkabilmektedir. Yazmaçların EAX, BX, R2, gibi özel adları vardır, ancak bu yazmaç adları bellek adresi değildir. Yazmaçlar RAM lara göre 2-10 kat daha hızlıdırlar.

Yazmaçlar, adres yazmaçları ve veri yazmaçları olarak 2 kümeye bölünebilirler. Bunların görevleri kısaca şunlardır:

- AMB'nin yaptığı hesaplamalar sırasında işlenenleri(sayıları) tutmak
- Hesaptan sonra hesabın sonucunu tutmak
- AMB'nin bellekte okuyacağı veya yazacağı yerin adresini tutmak
- MİB'nin bir sonra yürüteceği komutun adresini tutmak
- MİB'nin bellekten getirdiği komutu tutmak.

Bazı özel yazmaçların adları aşağıda verilmiştir.

- Bellekte okunacak veya yazılacak yerin adresini tutan ***Bellek Adres Yazmacı.***
- Bellekten getirilmiş olan veriyi tutan ***Veri yazmacı***
- Bir sonraki komutun adresini tutan, ***Komut sayacı***

- Bellekten getirilmiş olan komutu tutan ***Komut yazmacı***

Yazmaçların kullanımına bir örnek: (a + b) * (c – d) işleminin yazmaçlarda nasıl gerçekleştirildiğini görelim.

- a verisini R0 yazmacına yükle
- b verisini R1 yazmacına yükle
- R0 ve R1 yazmaçlarındaki verileri topla ve sonucu R2 yazmacına yükle
- c verisini R0 yazmacına yükle
- d verisini R1 yazmacına yükle
- R0 yazmacındaki veriden R1 yazmacındaki veriyi çıkar ve sonucu R3 yazmacına yükle.
- R2 yazmacındaki veri ile R3 yazmacındaki veriyi çarp ve sonucu R4 yazmacına yükle.

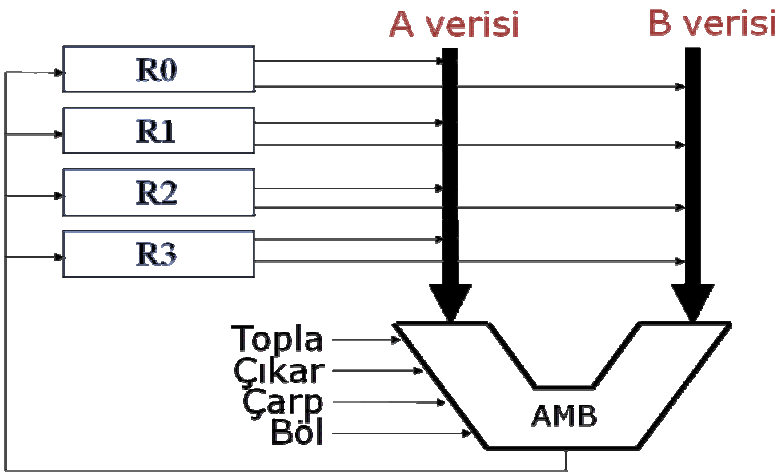
1.3.2. Aritmetik Mantık Birimi (AMB)

Aritmetik Mantık birimi aşağıdaki aritmetiksel ve mantıksal işlemleri yapar:

- +, -, * ve /
- VE (AND), VEYA (OR), DEĞİL (NOT)
- >, < ve =

AMB'nin bu işlemleri yaparken kullandığı veriler ve adresler, yazmaçlar üzerinde tutulur.

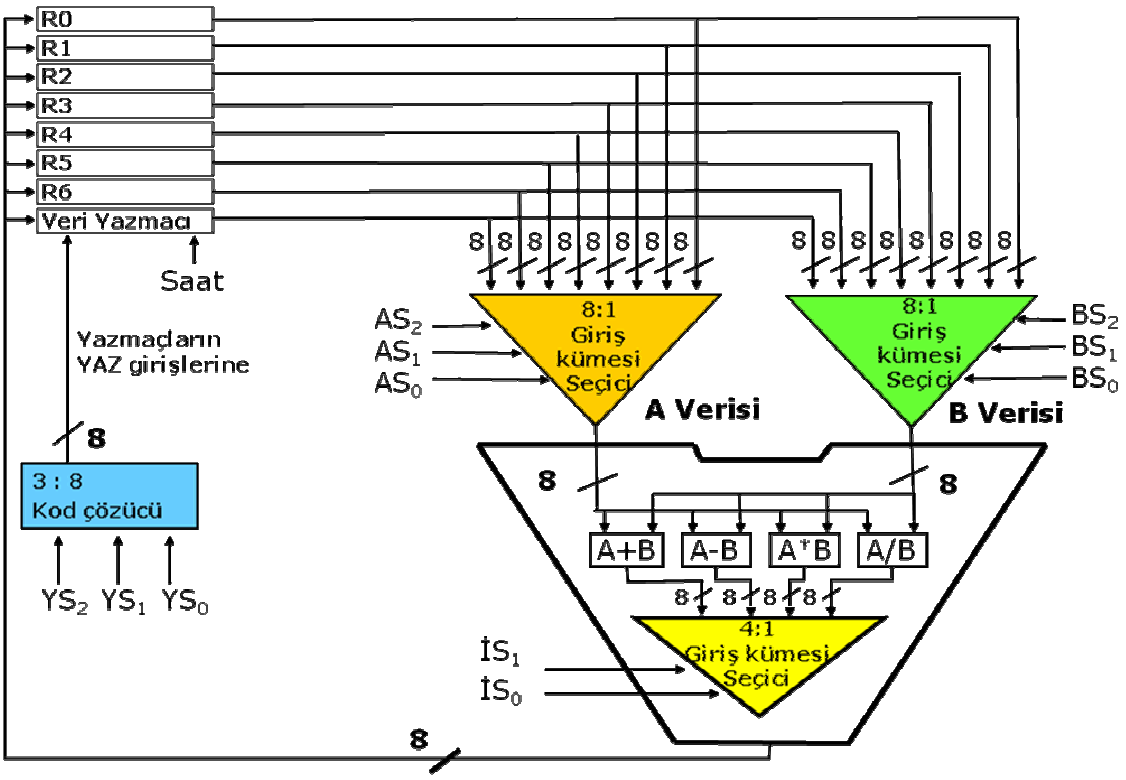
AMB'de her bir işlemi gerçekleştirmek için ayrı bir devre vardır. Bu işlemler, toplama, çıkarma, çarpma ve bölme gibi işlemler olabilir. Üzerine işlem uygulanacak veriler AMB nin sağ ve sol girişlerinden girer, içindeki mantık devrelerinden geçerek işlem çıkışlarını oluştururlar. Bu işlem çıkışları bir giriş seçici ile seçilerek, AMB nin çıkışına gönderilir. AMB nin basit bir öbek çizimi Şekil 1.5. te görülmektedir.



Şekil 1.5. AMB'nin basit şeması.

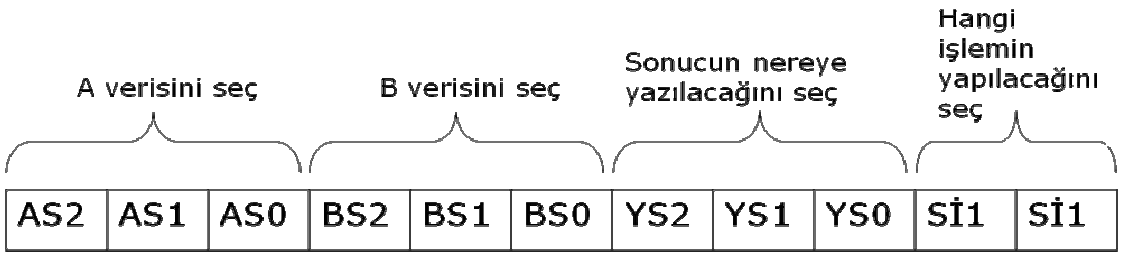
Aşağıdaki özelliklere sahip bir MİBnin, AMB'si ile yazmaçları arasında nasıl bir bağlantı olabileceğini inceleyelim.

- Sözboyu : 8
- İşlem yapabildiği yazmaçları:, R0,R1,R2,R3,R4,R5,R6 ve veri yazmacı. Bu yazmaçlar 8 bitlidir.
- Yapabildiği işlemler : Toplama, çıkarma, çarpma ve bölme.



Şekil 1.6. AMB ve yazmaçların ilişkisi.

Yukarıda şekil 1.6. da iç yapısı verilen AMB nin komut sözcüğünün yapısına bir örnek, Şekil 1.7. de verilmiştir(komut sözcüğü, komut yazmacında tutulur).



Şekil 1.7. Komut sözcüğünün yapısı.

1.3.3. Temel AMB işlemleri

Bir bilgisayarın aritmetik Mantık Birimi, verilerin saklanabileceği bir dizi yazmaçtan ve bu yazmaçlarda saklanan veriler üzerinde ve yazmaçlar arasında bazı işlemler

yapmaya yarayan bir dizi mantık devresinden oluşur. Bir yazmaçta saklanan veriler üzerinde şu işlemler yapılabilir:

- Yazmaç sıfırlanabilir. Yani tüm bitleri sıfır yapılabilir.
- Yazmacın içeriği 2'nin yada 1in tümleyeni biçimine dönüştürülebilir.
- Yazmacın içeriği sağa yada sola kaydırılabilir.
- Yazmacın içeriği 1 artırılabilir veya azaltılabilir.

Yazmaçlar arasında yapılan en önemli ve yaygın işlemler ise şunlardır:

- Bir yazmacın içeriği diğer bir yazmaca aktarılabilir.
- Bir yazmacın içeriği diğer bir yazmacın içeriğinden çıkarılabilir ya da diğer bir yazmacın içeriğine toplanabilir.
- AMB nin yaptığı işlemlerin çoğu bu yukarıda anlatılan iki işlem türünden oluşur. Çarpma bölme gibi daha karmaşık işlemler ise bu basit işlemlerin birkaçının bir dizi halinde ard arda uygulanması ile gerçekleştirilebilir.

AMB de kullanılan temel devrelerden birisi toplama devresidir. Toplama devreleri, aşağıdaki gibi 3 kümede incelenebilir.

- Çeyrek toplayıcı
- Yarım toplayıcı
- Tam toplayıcı

Tek bitli Çeyrek toplayıcının işlevi, ikilik düzendeki iki tane tek basamaklı işaretsiz tam sayıyı toplamaktır. Çeyrek toplayıcının iki tane sayı girişi vardır. Bir tane de sayı çıkışı vardır. Toplanacak sayılar sayı girişlerini oluşturur. Toplam ise sayı çıkışını oluşturur. Toplama işleminden elde çıkışı yoktur.

Çeyrek toplayıcının doğruluk Çizelgesi Çizelge 1.4. te verilmiştir. Mantıksal devresi ise Şekil 1.8. deki gibidir.

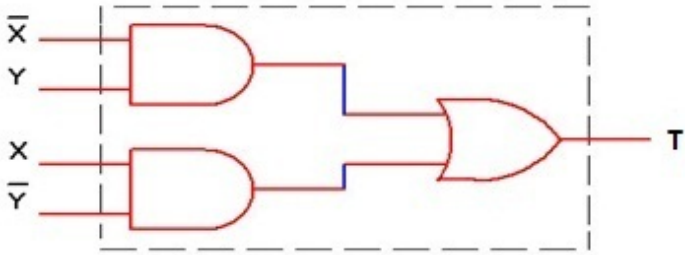
X: Birinci sayı girişi

Y: İkinci sayı girişi

T: Toplam

Çizelge 1.4. Çeyrek toplayıcının doğruluk Çizelgesi

GİRİŞ		ÇIKIŞ
X	Y	T
0	0	0
0	1	1
1	0	1
1	1	0



Şekil 1.8. Çeyrek toplayıcının devre şeması.

1.3.4. AMB Tek bitli yarım toplayıcı

Tek bitli Yarım toplayıcının işlevi, ikilik düzendeki iki tane tek basamaklı işaretsiz tam sayıyı toplamaktır. Yarım Toplayıcının iki tane sayı girişi vardır, iki tane de sayı çıkışı vardır. Toplanacak sayılar sayı girişlerini oluşturur. Sayı çıkışlarını ise toplam ve elde çıkışı oluşturur. Yarım toplayıcının çeyrek toplayıcıdan farkı elde çıkışı olmasıdır. Yarım Toplayıcının doğruluk çizelgesi Çizelge 1.5. te gösterilmiştir.

Çizelge 1.5. Yarım Toplayıcının doğruluk Çizelgesi

GİRİŞ		ÇIKIŞ	
X	Y	T	E
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

1.3.5. Tam toplayıcı

Birden çok sayıda basamağı olan ikilik iki tane sayıyı toplama işlemi, her bir basamak için ayrı bir toplayıcı kullanarak yapılabilir, ancak bunu yarım toplayıcılarla yapamayız, çünkü yarım toplayıcıların elde girişleri yoktur. Birden fazla basamağı toplarken her basamağın elde çıkışı olması gerektiği gibi en sağdaki basamak hariç diğer basamaklara elde girişi de olmalıdır. Bunun için tam toplayıcılar kullanmamız gerekir.

Tam toplayıcının işlevi, ikilik düzendeki iki tane tek basamaklı işaretsiz tam sayıyı toplamaktır. Tam Toplayıcının üç tane sayı girişi vardır, iki tane de sayı çıkışı vardır. Toplanacak sayılar ve giren elde, sayı girişlerini oluşturur. Toplam ve çıkan elde ise Sayı çıkışlarını oluşturur. Tam toplayıcının, Yarım toplayıcıdan farkı elde girişi olmasıdır.

Aşağıda tam toplayıcının doğruluk çizelgesi, Çizelge 1.6. da, öbek çizimi ise Şekil 1.9. da görülmektedir.

Çizelge 1.6. Tam toplayıcının doğruluk çizelgesi

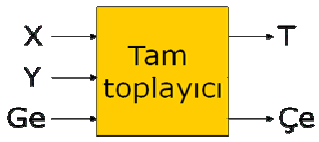
GİRİŞ			ÇIKIŞ	
X	Y	GE	T	ÇE
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Buradan Toplam ve Çe(çıkan elde) çıkışlarının ifadesi:

$$2.1.\quad T = \overline{X}.\overline{Y}.Ge + \overline{X}.Y.Ge + X.\overline{Y}.Ge$$

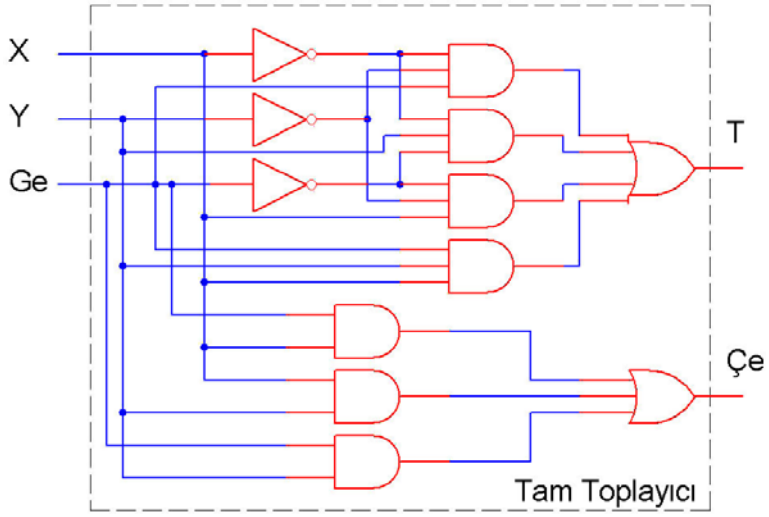
$$2.2\qquad 3E = XGe + XY + YGe$$

Şeklinde elde edilir.



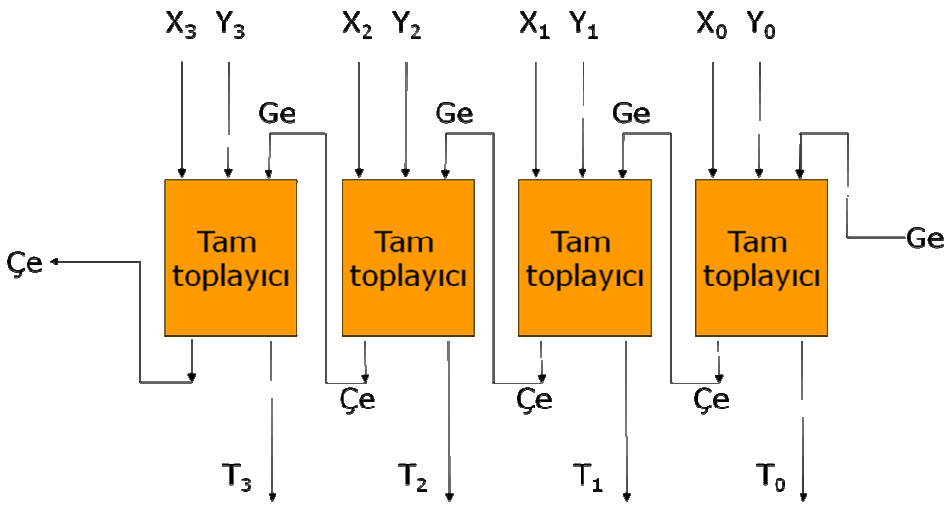
Şekil 1.9. Tam toplayıcının öbek çizimi(Block diagram).

Tam toplayıcının mantıksal kapılarla çizilmiş devre şeması Şekil 1.10. da görülmektedir.



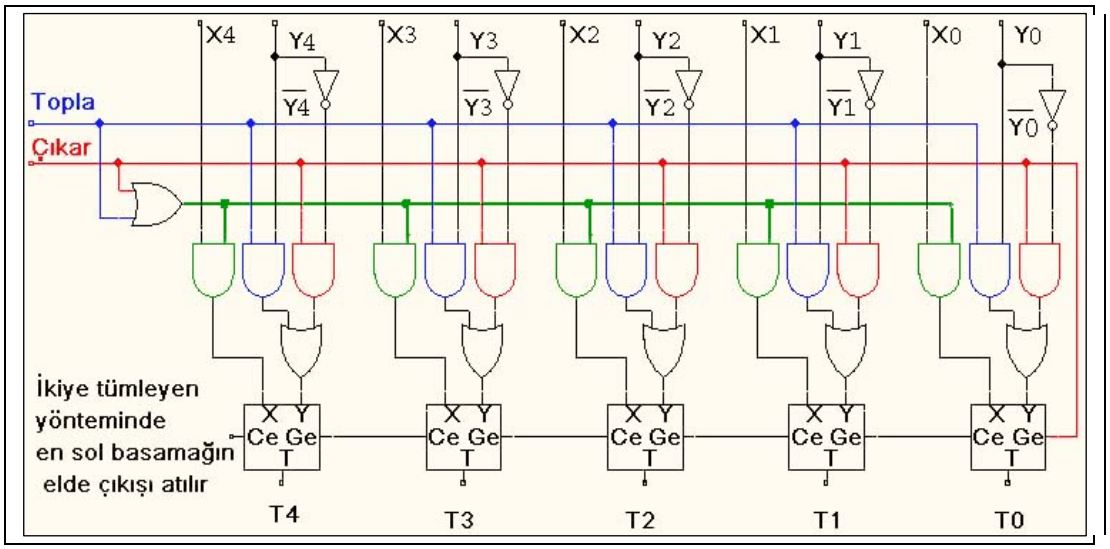
Şekil 1.10. Tam toplayıcının mantıksal devresi.

Dört bitli iki tane işaretsiz ikilik sayının tüm bitlerini paralel olarak toplayabilen, 4-bitli paralel bir toplama devresinin şeması, Şekil 1.11. de görülmektedir. Bu toplayıcının işlevi, ikilik düzendeki iki tane 4 basamaklı işaretsiz tam sayıyı toplamaktır. Toplayıcı, sayıların işaretlerine bakmaksızın yalnızca mutlak değerlerini toplamaktadır. Böyle bir toplayıcının devresi aşağıdaki şekilde görülmektedir. Burada X_3, X_2, X_1 ve X_0, toplanan sayı girişlerini göstermektedir. Y_3, Y_2, Y_1 ve Y_0 ise toplanılan sayı girişlerini göstermektedir. Ge giren eldeyi, Çe çıkan eldeyi, T ise toplamı göstermektedir.



Şekil 1.11. 4-Bitlik ikilik işaretsiz paralel tam toplayıcı.

Şekil 1.12. de ise işaretsiz beş-bitli iki sayıyı ikiye tümleyen yöntemi ile birbirinden çıkaran ve toplayan bir mantıksal devre görülmektedir.



Şekil 1.12. İkilik paralel beş-bitli ikiye tümleyen tam toplayıcı ve çıkarıcı.

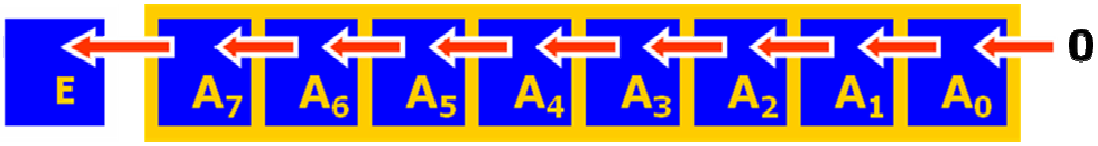
1.3.6. Taşma bayrağı

İşaretli sayılarla yapılan işlemler sonucunda ortaya çıkan sayı, gösterilebilir sayı sınırlarının dışında olursa, bunu belirtmek için taşma bayrağı kullanılır. Taşma bayrağının değeri bir Flip-Flop üzerinde bulunur. Eğer taşma bayrağının değeri 1 ise, işaretli sayılarla yapılan işlemler sonucunda ortaya çıkan sayı, gösterilebilir sayı sınırlarının dışında(taşma var) demektir. Eğer, taşma bayrağının değeri 0 ise, işaretli sayılarla yapılan işlemler sonucunda ortaya çıkan sayı, gösterilebilir sayı sınırlarının içinde(taşma yok) demektir. MİB, iki durumda taşma olduğuna karar verir ve taşma bayrağını 1 yapar. (1) İki artı sayı toplanır ve sonuç eksi çıkarsa. Yani, toplanan iki sayının da en büyük bitleri 0 olur ve sonucun en büyük biti 1 olursa. (2) İki eksi sayı toplanır ve sonuç artı çıkarsa. Yani, toplanan iki sayının da en büyük bitleri 1 olur ve sonucun en büyük biti 0 olursa.

1.3.7. Mantıksal sola kaydırma

Mantıksal sola kaydırma devresi, yazmaçta saklanmakta olan verinin her bir bitini belirtilen basamak kadar sola taşır. Bu işlemle yazmaçta bulunan sayı bir bit sola

kaydırıldığı zaman 2 ile çarpılmış olur. Şekil 1.13. te sekiz bitli ikilik bir sayıyı elde bayrağı üzerinden mantıksal olarak sola kaydırma işlemi yapan devrenin öbek çizimi görülmektedir.

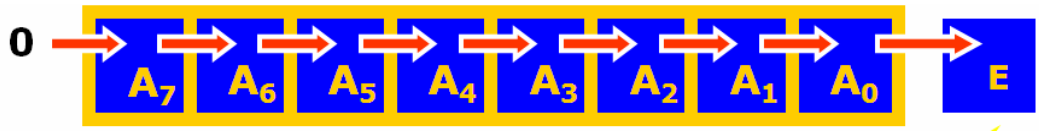


Şekil 1.13. Mantıksal sola kaydırma.

Örnek: 10111001 sayısını içeren bir yazmacın içeriği sola doğru 2 basamak kaydırıldıktan sonra 11100100 olur.

1.3.8. Mantıksal sağa kaydırma

Yazmaçta saklanmakta olan verinin her bir bitini belirtilen basamak kadar sağa taşır. Bu işlemle sayı 2 ye bölünmüş olur. Ancak sayı işaretli ise, sayının işareti değişebilir. Sayının işaretinin değişmemesi için aritmetiksel sağa kaydırma yapılmalıdır. Şekil 1.14. te sekiz bitli ikilik bir sayıyı elde bayrağı üzerinden mantıksal olarak sağa kaydırma işlemi yapan devrenin öbek çizimi görülmektedir.

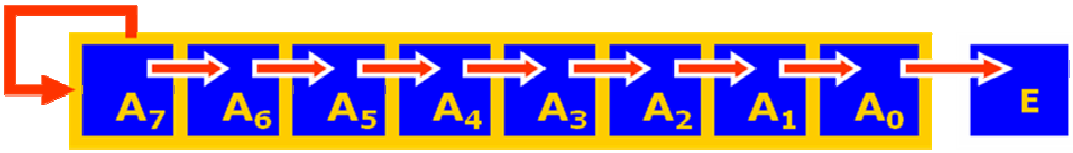


Şekil 1.14. Mantıksal sağa kaydırma.

Örnek: 10111001 sayısını içeren bir yazmacın içeriği sağa doğru 2 basamak kaydırıldıktan sonra 00101110 olur.

1.3.9. Aritmetiksel sağa kaydırma

Yazmaçta saklanmakta olan verinin her bir bitini belirtilen basamak kadar sağa taşır. Kaydırmadan sonra sayının sol tarafında oluşan boşluklar kaydırmadan önce sayının en solda bulunan bit ile doldurulur, böylece sayının işaret biti değişmemiş olur. Aritmetiksel olarak elde bayrağı üzerinden aritmetiksel olarak sağa kaydırma yapabilen sekiz bitli bir mantıksal devrenin öbek çizimi Şekil 1.2. te görülmektedir.

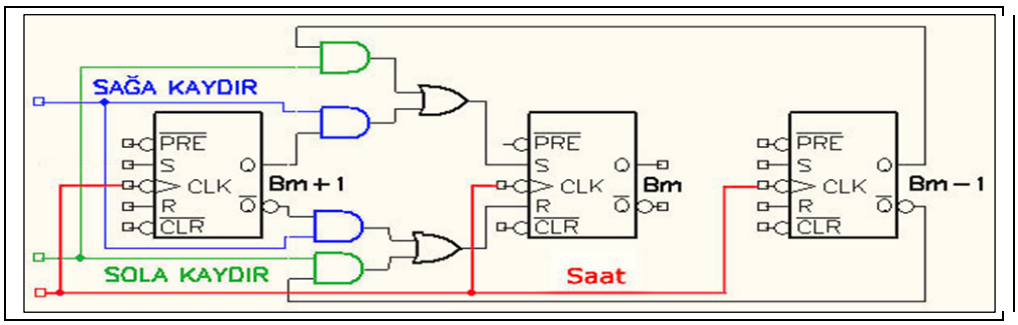


Şekil 1.15. Aritmetiksel sağa kaydırma.

Örnek: 10111000 sayısını içeren bir yazmacın içeriği sağa doğru, aritmetiksel olarak 1 basamak kaydırıldıktan sonra 11011100 olur.

Aritmetiksel sola kaydırma işlemi, mantıksal sola kaydırma işlemi ile aynıdır.

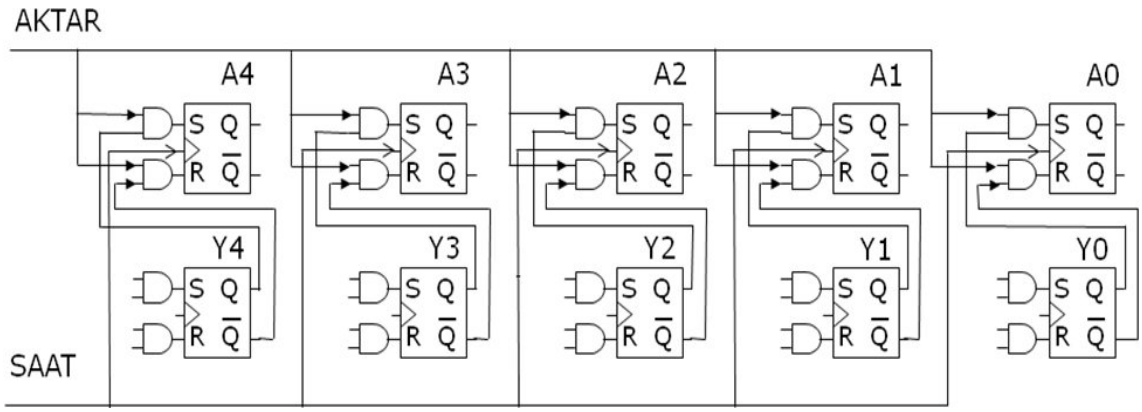
Sağa ve sola mantıksal olarak kaydırma yapabilen ve Flip Flop larla tasarlanmış bir mantıksal devre Şekil 1.16. te görülmektedir.



Şekil 1.16. Sağa ve sola tek basamak mantıksal kaydırma yapabilen mantıksal devrenin tek bir biti.

1.3.10. Aktarma işlemi

Aktarma, bilgisayarda önemli bir işlemdir. Örneğin aşağıda A ve Y yazmaçlarından oluşan aktarma devresini ele alalım. Burada A yazmacı, A_0, A_1, A_2, A_3 ve A_4 yazmaçlarından, Y yazmacı ise, Y_0, Y_1, Y_2, Y_3 ve Y_4 yazmaçlarından oluşmaktadır. Aktarma işlemi şu şekilde gerçekleşir: AKTAR hattına 1 verildiği zaman, saat sinyalın gelmesi ile, Y yazmacının içeriği, A yazmacına kopyalanır. Beş bitli iki yazmaç arasında aktarma işlemi yapabilen bir aktarma devresinin şeması Şekil 1.17 de görülmektedir.



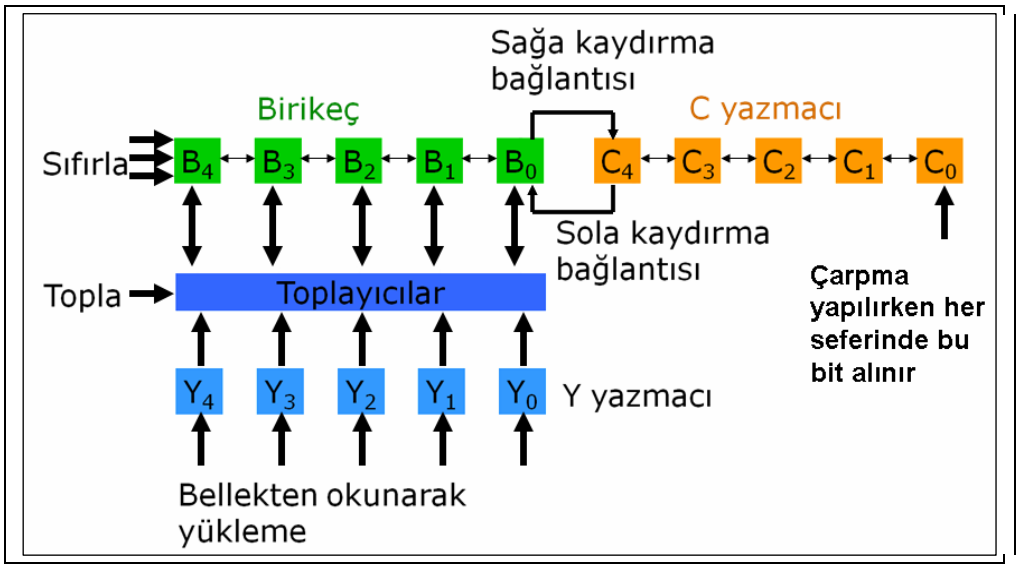
Şekil 1.17. Aktarma işlemi yapan mantıksal devre.

1.3.11. Çarpma işlemi

İki sayıyı çarpma işlemi, çarpılan sayıyı çarpan sayı kadar kendi kendisi ile toplayarak yapılabilir. Bu yöntemle çarpma yaparken, çarpılan sayının kendi kendisi ile kaç kere toplandığının hesabını tutmak için çarpan sayı bir yazmaca yüklenir ve her toplama işleminden sonra yazmaçtaki sayı bir azaltılır. Bu yazmaçtaki sayı sıfıra gelene kadar çarpılan sayı kendi kendisine toplanır.

Bu yavaş bir çarpma tekniğidir. Sayısal bilgisayarlarda çarpma işlemi yapmak için çok daha hızlı algoritmalar kullanılır. Ancak hızlı işleminde bir bedeli vardır. Hızlı işlem yapan devreler daha pahalıdır. Hızlı işlem yapan bilgisayar, pahalı bilgisayar demektir. Bilgisayarda işlem yapmak için hangi tekniğin kullanılacağına karar vermek için hız ile ucuzluk arasında bir tercih yapmamız gerekir. Daha hızlı çarpma yapmak için bir

algoritma geliştirilebilir. Şekil 1.18 de toplayarak çarpmaya göre daha hızlı çarpma yapan, algoritmik bir çarpma devresi görülmektedir [9].



Şekil 1.18. Algoritmik çarpma devresi.

1.4. 6800 8-bit Mikroişlemcisi

Motorola 6800 8-bit bir işlemci olup Intel 8080'den az bir süre sonra 1974 yılında piyasaya sürülmüştür. 78 komuttan oluşan bir komut kümesine sahiptir ve Indeks yazmacı kullanan ilk mikroişlemcidir. Motorola 6800 işlemcinin programlama modeli aşağıda verilmiştir

A - Accumulator A

B - Accumulator B

X - Index register (indeks yazmacı)

PC - Program Counter (program sayacı)

SP - Stack Pointer (yığıt göstergeci)

CCR - Conditional Code Register: Half carry, Interrupt mask, Negative, Zero, oVerflow and Carry

6800 işlemci standart bir 8-bit ikinin-tümleyeni tabanlı bir işlemcidir. 8-bit'lik bayt'ları

ve 16-bit adres ile 64KB belleği destekleyen bir işlemcidir. Ayrıca 8080 gibi kod ROMda (yalnız okunabilir bellek) saklanabilir.

6800 işlemcide 16-bit bir yığıt göstergeci vardır, bu nedenle yığıt bellekte herhangi bir yerde olabilir ve bellek boyuna kadar genişleyebilir. 6502 işlemciyle karşılaştırılınca 6502'nin sadece 8-bit'lik bir göstergeci vardır ve yığıt 256-511 adreslerine sabitlenmiştir.

8080 gibi 6800 de "carry flag" ve ADC (add with carry, elde ile birlikte topla) komutlarını kullanarak 1 bayt'dan daha büyük sayıları toplayabilir. Benzer şekilde SBC (subtract with carry- elde ile çıkar) komutu da vardır. Ondalıklı aritmetik için de 8080'e benzer şekilde DAA (decimal adjust accumulator- akümülatörü ondalıklı ayarla) komutu ile 2 ye tümleyen biçimindeki bir sonucu 2 adet paketlenmiş ondalık rakam şekline çevirebilir. 6502'den farklı olarak, 6800'de "carry" kullanmadan toplama ve çıkarma yapan (ADD ve SUB) komutları vardır.

Aritmetik komutlar 2'nin tümleyeni bayrakları değiştirirler: İşaret(sign), Sıfır(zero), taşma (overflow) ve elde (carry). 8080 ve 6502'den farklı olarak, 6800'de bütün akış dallandırma (branch) komutları, PDP-11'dekilerle birebir aynıdır, hem işaretli hem de işaretsiz değerleri karşılaştırabilir. Eğer "i" ve "j"yi karşılaştırmak istersek önce "LDAA i" komutu ile "i"nin değeri akümülatöre yüklenir sonra da "CMPA j" komutu ile "j"nin değeriyle karşılaştırılır ve karşılaştırma sonucuna göre program akışında dallanma yapılır.

PDP-11, 6502 ve 8080'den farklı olarak 6800 mikroişlemciler IBM 360 tarafından da kullanılan "büyük önde" (big-endian) bayt sıralaması kullanmaktadır.

PDP-11 ve 6502'nin tersine, 8080 gibi, 6800 de "borrow carry" kullanmakta yani çıkarma işlemi sırasında gerektiğinde "carry" bayrağı seçilmektedir.

8080 ve PDP-8'de olduğu gibi, ancak PDP-11'in tersine, 6800'de özel Giriş/Çıkış

komutları bulunmamaktadır. Bunun yerine Giriş/Çıkış aygıtları bellekle aynı adres uzayını paylaşmaktadırlar.

8080'in tersine, 6800 indeksleme yeteneğine sahipti, böylece veri yapılarını doğrudan desteklemekteydi. Bir veri yapısının başlangıç adresi indeks yazmacına yüklenmekte, ve sonra da 8-bit'lik bir işaretsiz "kayma" (offset) bilgisi indeksleme komutlarına eklenerek veri yapısından farklı elemanlar okunmaktadır.

6800 işlemcinin 4 adresleme kipi vardır: anında (immediate), indeksli (indexed), uzatılmış (extended), ve doğrudan (direct/zero page). Sıfır sayfası kipi daha hızlıdır çünkü sadece belleğin ilk 256 bayt'lık kısmına erişir ve daha kısa komutlar kullanılmasına izin verir. Sıfır sayfası yazmaçlar kümesinin genişletilmesine benzetilebilir.

Komut kısaltmaları PDP-11'e çok benzemektedir. ARM da dahil olmak üzere birçok mikroişlemcinin komut kısaltmları, 6800 kısaltmalarına oldukça benzerdir. 6800 veri yolu 8080'e benzerdir ve bir çevre birim TD(Tümleşik Devre)'leri ailesinin merkezindedir. Veri yolunu kullanan birimler 3 kontrol sinyaline ve adres ve veri yollarına gerek duyarlar.

- Φ2 veya E: bir saat çevrimi ardarda gelen iki düşen E ucu olarak tanımlanmaktadır.
- R/W: eğer 1 ise okuma çevrimidir, eğer 0 ise yazma çevrimidir.
- CS(chip select): eğer 0 ise yonga seçilir. Bunun için adres yolundaki AS (address select) veya VMA(valid memory address) kullanılır.

1.4.1. 6800 8-bit işlemcisinin zayıf yönleri

Bu zayıf yönler daha sonra geliştirilen 6809 and 6811 işlemcilerde giderilmiştir. (1) Sadece bir göstergeç var olması. Göstergeç olarak yığıt götergeci kullanılabilmesine rağmen bunun için kesmelerin döngüler sırasında engellenmesi gerekmektedir. 6800'ün

en önemli rakibi olan 8080'de ise tam 3 göstergeç vardır (ancak indeksleme yoktur).

Yığıt komutları ekleme sırasında sonra-azalt, ve yığıttan çıkarma sırasında da önce-artır yöntemini kullanmaktadır, oysa bunların yerine sırasıyla önce-azalt ve sonra-artır kullanılmalıdır (böylece SP her durumda yığıtın en üstündeki elemanı gösterecektir). Bu nedenle yığıt göstergeci indeks yazmacına transfer edildiğinde, yığıtın en üstündeki elemanın indeksi 0 yerine 1 olmaktadır.

İndex yazmacı doğrudan yığıta eklenememekte ve yığıttan alınamamaktadır. Akümülatör yazmacı ve indeks yazmaçları farklı yerlerdedir ve bu iki yazmacı birbirine transfer edecek bir komut yoktur. Bir başka örnek akümülatörü veya bir sabiti bir indeks yazmacına eklemek için bir yol yoktur. Daha sonra geliştirilen işlemcilerde ABX (add B to X, B'yi X'e topla) ve LEA (load effective address, etkin adresi yükle) komutları eklenerek bu eksiklik giderilmiştir.

CPX (compare X) komutu "carry" bayrağını etkilememektedir, bu nedenle indeks yazmacı ile doğrudan büyüklük karşılaştırması yapılamamaktadır.

DAA (decimal adjust) komutu yalnızca toplama komutundan sonra kullanılabilmektedir, çıkarmadan sonra kullanılamamaktadır. Ondalıklı çıkarma yapabilmek için 9'un tümleyeni alınmalı ve sonra toplama yapılmalıdır. 6502'de ise ondalık kipi biti bulunmaktadır. Z80 işlemcide çıkarma biti status bayrağında bulunmaktadır ve bu bit 1 iken toplama ve çıkarma komutları paketlenmiş ondalık sayılar üzerinde çalışmaktadır. 8086'da ise DAS (çıkarma için ondalık ayarla) komutu vardır.

2. BÖLÜM

BİR MİKROİŞLEMCİ, MAKİNE DİLİ VE EMÜLATÖRÜNÜN TASARIMI

Bu kısımda basit bir mikro işlemci tasarımı yapılmakta, ayrıca bu dile uygun bir makine dilinin ve simülatörünün ayrıntıları verilmektedir. Önerilen makine dili 6800 mikroişlemcisinin komutlarının en çok kullanılan küçük bir alt kümesinden oluşmuştur. Böylece hem öğrencilere bir makine dilinin komutlarının çalışması pratik bir şekilde gösterilmiş, hem de 6800 işlemcisine ve makine diline bir ilk tanıtım yapılmış olacaktır.

2.1. Makine dili komutları

En sık kullanılan makine dili komutları akümülatör yazmacına değer yükleyen ve akümülatördeki değerleri işleyen (toplama, belleğe saklama gibi) komutlardır. Bu tezde tasarlanan işlemcinin bir adet 8-bit akümülatörü (“A”), bir adet 16 bit indeks yazmacı (IX), ve 16-bit program sayacı (PC) vardır.

LDA <değer> – “A” akümülatörüne bir değer yükle
STA <bellek adresi> - “A” akümülatörünün değerini belleğe kopyala
ADA <değer> veya <bellek adresi>

Diğer önemli komut kümesi program akışını dallandırma (“branch”) komutlarıdır. Bu komutlarda “göreceli” adresleme kullanılmakta olup, “-128” ile “+127” arasında bir değer program sayacı (PC) yazmacına eklenerek, program akışını, elde edilen bu yeni adresteki komut, bir sonra çalıştırılacak komut olacak şekilde değiştirmek mümkündür. Böylece değişik koşullar altında programın değişik bölümlerinin çalıştırılması ve farklı işlemler yapılması gerçekleştirilmektedir. Bu komutların yeterli işlevselliği sağlayacak

küçük bir alt kümesi emülatör tarafından tanınmakta ve emüle edilmektedir, ilerde bu komutların hepsini çalıştıracak şekilde makine dili zenginleştirilebilir. Bu komutların, bu tezde tasarlanmış olan emülatör tarafın emüle edilebilen alt kümesi aşağıda verilmiştir.

BRZ	eğer "sıfır" ise
BEQ	eğer "eşit" ise
BLT	Eğer "küçüktür" sıfır ise
BGT	Eğer "büyüktür" sıfır ise

Aritmetik ve mantık kaydırma komutları akümülatörün içindeki değeri sağa veya sola kaydırma ve döndürme işlemlerinin gerçekleştirmesi için kullanılırlar.

ASL	Aritmetik sola kaydır
ASR	Aritmetik sağa kaydır
LSR	Mantıksal sağa kaydır
ROR	Sağa döndür
ROL	Sola döndür

Diğer bir önemli komut kümesi de STATUS (durum) yazmacında bulunan C (carry/elde), Z (zero/sıfır), N (negatif) gibi bayrak değerlerini değiştiren komutlardır.

SEC	C ← 1	SEZ	Z ← 1
CLC	C ← 0	CLZ	Z ← 0
SEN	N ← 1	CLN	N ← 0

2.2. Makine dili adresleme gösterimi

6800 işlemcisinde dört adresleme kipi bulunmaktadır. Bunlar aşağıda makine dili gösterimleriyle (LDA komutu için) birlikte verilmiştir.

- Doğrudan (immediate): LDA#
- Kısaltılmış (direct): LDA.
- Uzatılmış (extended): LDA$
- İndeksli (indexed): LDA,

Bu komutlardan sadece "uzatılmış" adresleme bellekte komuttan sonra 2 bayt adres bulunmasını beklemektedir. Fazladan bir bayt kullanılması bellekten daha fazla verinin işlemciye yüklenmesini gerektirdiği için 1 işlemci çevrimi daha uzun sürmektedir. Bu nedenle zorunlu olmadıkça diğer adresleme yöntemlerinin tercih edilmesi önerilebilir.

2.3. Makine dili tanımlaması

Aşağıda **Şekil 2.1. de** assembler dilinde yazılmış küçük bir program verilmiştir. "ORG" (origin) komutu programın bellekte başlangıç adresini vermektedir. "DB" (define byte) komutu ise bellekte önceden tanımlanmış değerlerin yerleştirilmesine izin vermektedir. "END" komutu ise programın çalışmasını sona erdirmektedir. Programın çalışması bittiğinde bellekte "100" adresinde toplam değer bulunmaktadır.

```
ORG 101
DB 1
DB 5
DB 30
DB 14
LDA. 101
ADD. 102
ADD. 103
ADD. 104
STA. 100
END
LIST 100
EXEC 105
```

Şekil 2.1. Bellekteki 3 sayıyı toplayan örnek program.

Bu programın çıktısı ise aşağıdaki **Şekil 2.2.** de görülmektedir.

Tasarlanan assembler dilinin özel komutları olan ORG ve DB aşağıda ayrıntılı olarak açıklanmaktadır. Bu komutlar makine dilinin bir parçası değildirler, ancak pratik bir program geliştirebilmek için gerekli olan belleğe veri yerleştirilmesi ve komutların bellekte nereye yerleştirileceği konularında kullanıcıya kontrol mekanizmaları sağlamaktadırlar.

```
0: ORG 101
101: DB 1
102: DB 5
103: DB 30
104: DB 14
105: 150 LDA. 101
107: 155 ADD. 102
109: 155 ADD. 103
111: 155 ADD. 104
113: 151 STA. 100
MEM[100-119]:0 1 5 30 14 150 101 155 102 155 103 155 104 151 100
0 0 0 0 0
105:LDA.101 PC=107 A=1 B=0 X=0 SP=0 C:0 Z:0 N:0
107:ADD.102 PC=109 A=6 B=0 X=0 SP=0 C:0 Z:0 N:0
109:ADD.103 PC=111 A=36 B=0 X=0 SP=0 C:0 Z:0 N:0
111:ADD.104 PC=113 A=50 B=0 X=0 SP=0 C:0 Z:0 N:0
113:STA.100 PC=115 A=50 B=0 X=0 SP=0 C:0 Z:0 N:0
STOPPED AT:115
MEM[100-119]:50 1 5 30 14 150 101 155 102 155 103 155 104 151
100 0 0 0 0 0
```

Şekil 2.2. Bellekteki 3 sayıyı toplayan programın çalışması ve çıktısı.

2.3.1. ORG komutu

Programın bellekte istenen adrese yerleştirilebilmesi önemli bir özelliktir. Bunu sağlamak için "ORG" yönerge komutu kullanılır. Böylece "program sayacı" verilen değerle ayarlanacak ve üretilecek makine dili komutları bu adrese yerleştirilecektir. Örnek olarak "ORG 100" yönergesi makine dili komutlarının belleğe 100'üncü karakter adresinden başlanarak yerleştirilmesini sağlayacaktır.

2.3.2. LIST komutu

Bu komut ile bellekteki belli bir adreste bulunan değerler ekranda gösterilmektedir. Örneğin komut olarak "LIST 100" verilirse bellekte 100 ile 119 arasındaki bellek adreslerinde bulunan veriler ekranda gösterilecektir. Bu komutla bellekte istenen komutların ve verilerin girildiği denetlenebilmektedir.

2.3.3. EXEC komutu

Bu komut ile bellekte istenen adres PC yazmacına yüklenerek komutlar çalıştırılmaya başlanmaktadır. Her bir komut çalıştırıldıkça işlemcinin içinde bulunan bütün yazmaçların ve durum bayrağının (status flag) değerleri kolayca anlaşılabilecek bir şekilde ekrana yazdırılmaktadır. Bu şekilde öğrenciler hem çalıştırılan komutu görecek hem de bu komutun işlemci yazmaçları üzerindeki etkisini görerek makine dilini daha kolay bir şekilde öğrenebilecektir.

2.3.4. DB (define byte) komutu

Bu komutla bellekte o anda bulunulan adrese tanımlanan değer yüklenecektir. Bu değerlerin 0-255 aralığında olması gerekir. Örnek olarak belleğe "SELIM" değerlerini yerleştirmek için aşağıda, Şekil 2.3. teki komutlar girilmelidir.

```
ORG 100
DB 83
DB 69
DB 76
DB 73
DB 77
END
```

Şekil 2.3. DB ile bellekte ilk değer olarak "SELIM" atanması.

Aynı komutların daha okunabilir olması açısından
Şekil 2.4. te görüldüğü gibi girmek de mümkündür. Burada doğrudan karakter kodları kullanıldığı için kullanıcı ve öğrenciler için büyük kolaylık olacak, ayrıca öğrenciler de kolay bir şekilde karaterlerin ASCII kodlarıyla tanışmış olacaklardır.

```
100: ORG 100
101: DB S
102: DB E
103: DB L
104: DB I
105: DB M
MEM[100-119]:83 69 76 73 77 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Şekil 2.4. Büyük karakter sabitlerle değer atama.

Bu kodun çalıştırılması sonucunda elde edilen bellek çıktısı aşağıdaki gibidir:

```
MEM[100-119]:83 69 76 73 77 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Burada görüldüğü üzere ‘S’ harfine karşılık olarak ‘83’ bellekte 100 adresli pozisyona yerleştirilmiştir. Diğer rakamlar da sırasıyla ‘E’,’L’,’I’, ve ‘M’ harflerine karşı gelmektedir. Bu harflerin yerine alfabenin küçük harfleri ‘s’, ‘e’, ‘l’,’i’,’m’ şeklinde kullanılırsa bellekteki değerler sekil 2.5 teki gibi olacaktır.

```
100: ORG 100
101: DB s
102: DB e
103: DB l
104: DB i
105: DB m
MEM[100-119]:115 101 108 105 109 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Şekil 2.5. Küçük karakter sabitlerle değer atama.

Bu örnekte ‘S’ için ASCII kodu 83 iken ‘s’ için ASCII kodunun 115 olduğu görülmektedir.

2.3.5. LDA komutu

Bu komut en sık kullanılan komutlardan biri olan A akümülatörünü yükle (load accumulator A) komutudur. 6800 işlemcisinde "A" ve "B" adlarında iki akümülatör vardır, ancak daha basit bir tasarım olarak bu emülatörde LDA komutu doğrudan A akümülatörünü işlemek üzere tanımlıdır. İlerde "B" akümülatörü de işlenen komut setine dahil edilmek istenirse o zaman bu komut "LDAA" ve "LDAB" şeklinde değiştirilerek daha genel bir formata çevrilebilecektir.

2.3.6. ADD komutu

Bu komut ile "A" akümülatörüne bir değer eklenmektedir. Bu komutun doğrudan değer içeren (immediate), birinci bellek bloğunda bir adresteki değere referans veren, veya uzatılmış (extended) 16 bit adreslerle belleğin bütünü içinde bir adrese erişebilir. Toplama sonucu gene "A" akümülatörü içinde bulunacaktır.

LDA ve ADD komutları içeren program parçası ile bu kod çalıştırıldığında elde edilen çıktı, Şekil 2.6. da verilmiştir:

ORG 100 DB 78 LDA# 50 ADD. 100 LIST 100	134 LDA(2) PC=101 A=50 B=0 X=0 155 ADD(2) PC=103 A=128 B=0 X=0 MEM[100-119]:78 134 50 155 100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Şekil 2.6. ADD komutu ve çıktısı.

Çıktıda görülen ilk satır olan "134 LDA(2)" ifadesi "LDA" komutunun makine kodu karşılığı "134"tür, "2" ise bu komutun 2 bayt uzunlukta olduğunu gösterir. "155 ADD(2)" ise "ADD" komutunun "direct" erişim modundaki karşılığı olan "155"i

göstermektedir ve “direct” modunda adresler 1 bayt uzunluğunda olduğu için toplam komut uzunluğu gene “2” bayt olmaktadır. Bu işlemin gerçekleştirilmesi sonucunda “LDA# 50” ile 50 değeri yüklenmiş olan “A” akümülatörüne “100” adresinde “DB 78” komutu ile yüklenmiş olan “78” değeri “100” adresinden okunarak eklenmektedir. Bu toplama işleminin sonucu olan “A=128” çıktısı da emülatör tarafından ekrana basılmıştır.

“LIST 100” ile bellekteki değerler “100” adresinde itibaren yazdırıldığında “100” adresinde “DB 78” komutu ile bu adrese yerleştirilen “78” değeri en baştadır. “101” ve “102” adreslerinde sırasıyla “134” ve “50” görülmekte olup “134” “LDA#” komutuna, “50” ise bu komutun bellekteki argümanına denk gelmektedir. “103” ve “104” adreslerinde görünen “155” ve “100” değerleri ise sırasıyla “ADD.” ve toplanacak değerin bellekteki adresi olarak “100” görülmektedir.

2.3.7. STA komutu

Bu komut “A” akümülatöründe bulunan değerin bellekte bir adrese yazılmasını gerçekleştirir. Böyle “A” akümülatörü bellekten başka değerleri yükleyip işlemek üzere boşaltılmış olur. Yukarıdaki örnekte toplanan değer bellekte “99” adresinde saklanmak istenirse o zaman toplama işleminden sonra “STA. 99” komutu ile “A” akümülatörünün değeri bellekte “99” adresinde saklanabilir ve bu değerin bellekte istenen adrese yazılmış olduğu da “LIST 99” ile ekrana yazdırılabilir. Şekil 2.7. deki çıktıda bellekte “99” adresinde “128” değerinin başarıyla saklandığı görülmektedir.

ORG 100	134 LDA(2)
DB 78	PC=101 A=50 B=0 X=0
LDA# 50	155 ADD(2)
ADD. 100	PC=103 A=128 B=0 X=0
STA. 99	151 STA(2)
LIST 99	PC=105 A=128 B=0 X=0
END	MEM[99-118]:128 78 134 50 155 100 151 99 0 0 0 0 0 0
	0 0 0 0 0 0

Şekil 2.7. STA komutu ve çıktısı.

2.3.8. AND komutu

Bu komut "A" akümülatöründe bulunan veri ile bellekte bulunan başka bir verinin bitlerine mantıksal VE işleminin uygulanmasını sağlar. Bu komutun doğrudan parametre alan, DIRECT ve diğer adresleme kiplerinde çalışan kipleri vardır.

2.3.9. TAB ve TBA komutları

LDA komutu ile A akümülatörüne değer yüklenebilmektedir ancak B akümülatörü için bir yükleme ve depolama komutu yoktur. Bunların yerine TAB (A yazmacını B'ye transfer et) ve TBA (B yazmacını A'ya transfer et) komutları kullanılarak A yazmacına önce yüklenen değer B akümülatörüne aktarılmakta ve B yazmacına değer yüklenmiş olmaktadır. B yazmacındaki değeri belleğe yazmak için de önce bu değer A yazmacına aktarılmakta sonra da STA komutu ile bu değer belleğe yazılmaktadır. Böylece A akümülatörüne yardımcı olarak kullanılabilecek ikinci bir B akümülatörü elde edilmektedir.

2.3.10. ABA komutu

A akümülatörüne doğruda bellekten bir değer toplanabilmektedir ancak B akümülatörü için bu komut yoktur, onun yerine B yazmacının değerini A yazmacının üstüne toplayan "ABA" komutu vardır ve toplama sonucu A yazmacına saklanmaktadır.

2.3.11. B yazmacını işleyen diğer komutlar

Artırma (ICB), azaltma (DCB), sola/sağa döndür (ROL/ROR B) komutları kullanılarak B yazmacı işlenebilmektedir.

2.4. Geliştirilen C++ yazılımında ana veri yapısı

Geliştirilen C++ yazılımında ana veri yapısı şu dizidir:

```
struct IS {
 int opcode;  // operatörün sayısal kodu
 char MNE[4]; // anlaması kolay 3 karakterlik operatör kod
 int NOB;     // komutun bellekte kapladığı toplam byte sayısı
 int ADM;     // komutun kullandığı adresleme modu
 int CYC;     // komutun gerçekleştirilmesi için gereken çevrim sayısı
 int EXP;     // açıklama metni
};
```

"opcode" değerinin tanımlı diğer komutlardan farklı olması gereklidir. Bu değerin 1 byte içine sığan 0-255 arasında bir değer olduğu varsayılmaktadır.

"MNE" değeri de "ADD", "SUB" şeklinde 3 karakteri geçmeyen bir koddur ve operasyon kodu gibi diğer tanımlı komulardan farklı olması ve yapılan işlevi kolayca akla getirecek bir şekilde seçilmesi gerekmektedir.

"NOB" değeri program yazmacının bir sonraki komutun adresini bulmak için ne kadar artırılacağını belirtir. 1 byte operatör kodu ile varsa komut parametrelerinin bellekte tuttuğu toplam byte sayısını verir.

"ADM" değeri komutun kullanacağı adresleme modunu belirtmekte ve emülatörün komutu çalıştırması için yardımcı bilgi olarak kullanılmaktadır. Örneğin "IMMEDIATE"adresleme modu kullanılıyorsa o zaman komutun hemen yanındaki byte'da bir parametresinin bulunduğu

anlaşılmaktadır.

"CYC" değeri emülatör tarafından şu anda kullanılmaktadır. Ancak ilerde bir programın gerçek zamanlı olarak çalışma süresinin izlenmesinde kullanılabilecektir. Özellikle belirli bir süre içinde tamalanması gereken komutların kaç çevrim sürdüğünün yazılımcı tarafından hesaplanması gerekecektir.

```
class Oper6800 {
 int A, B, PC, X, SP;
 int FH, FN, FI, FC, FZ, FV;
 unsigned char FLAGS;
 unsigned char MEM[65000];
...
};
```

Bu sınıf içinde işlemcinin durumu depolanmaktadır. A, B, PC, X, ve SP işlemcinin temel yazmaçlarının değerlerini saklamaktadır. İşlenen komutlardan sonra işlemci bayraklarının saklanması ve bu bayrak değerlerini kullanarak diğer komutların seçimli olarak çalıştırılması sağlanmaktadır.

MEM[65000] dizisi işlemcinin belleği olarak kullanılmaktadır. Assembler Komutları işlenerek operatör kodları ve parametre değerleri bulunarak belleğe yerleştirilmekte ve bellekten daha sonra okunarak çalıştırılmaktadır.

3. BÖLÜM

DENEYSEL SONUÇLAR

Bu bölümde geliştirilen emülatörün yeteneklerini göstermek ve örnek uygulamalar elde etmek üzere yapılan deneysel çalışmalar ve elde edilen sonuçlar anlatılmaktadır. İlk olarak 3 adet sayıyı toplayan bir program aşağıda verilmiş ve ayrıntılı olarak anlatılmaktadır.

3.1. Örnek bir toplama programı

```
ORG 100
DB 0
DB 5
DB 30
DB 14
LDA# 0
ADD. 101
ADD. 102
ADD. 103
STA. 100
LIST 100
END
```

Şekil 3.1. Örnek bir toplama programı.

Örnek bir toplama programını inceleyelim, Şekil 3.1. de gösterilmiş olan bu ilk programın en başında bulunan “ORG 100” komutu programın bellekte 100 numaralı

bayt'dan başlayarak yerleştirileceğini göstermektedir. İlk 4 bayt'ın bellekteki değerleri daha sonra "DB" komutları ile verilmektedir. Böylece bellekte 100, 101, 102, ve 103 adresli baytlara sırasıyla 0, 5, 30, ve 14 yerleştirilmektedir. Daha sonra "LDA# 0" komutu ile "A" akümülatörüne sıfır değeri yüklenmekte ve "A" akümülatörüne 101, 102, ve 103 numaralı bellek adreslerindeki baytlarda bulunan değerler toplamak için "ADD. 101", "ADD. 102", ve "ADD. 103" komutları kullanılmaktadır. Bu "ADD" komutlarının sonundaki "." ifadesi bellekten "DIRECT" (yani sadece 1 bayt'lık adresler kullanarak sadece 0-255 numaralı bellek adresleri arasındaki değerlere erişmek için kullanılan kompakt bir adresleme kipi) adresleme yapılacağını belirtmektedir. Bu komutların çalıştırılmasının sonucu aşağıda Şekil 3.2. de verilmiştir.

```
0: ORG 100
100: DB 0
101: DB 5
102: DB 30
103: DB 14
104: 134 LDA# 0 PC=106 A=0 B=0 X=0 C:0 Z:1 N:0
106: 155 ADD. 101 PC=108 A=5 B=0 X=0 C:0 Z:0 N:0
108: 155 ADD. 102 PC=110 A=35 B=0 X=0 C:0 Z:0 N:0
110: 155 ADD. 103 PC=112 A=49 B=0 X=0 C:0 Z:0 N:0
112: 151 STA. 100 PC=114 A=49 B=0 X=0 C:0 Z:0 N:0
MEM[100-119]:49 5 30 14 134 0 155 101 155 102 155 103 151 100 0 0 0 0 0 0
```

Şekil 3.2. Toplama programının çalışması ve çıktısı.

Görüldüğü üzere program çalışması bittiğinde 100 nolu bellek adresinde "49" değeri bulunmaktadır. Bunun nedeni "STA. 100" komutu ile "5+30+14"in A akümülatöründe saklanan toplam değeri olan "49" sayısının "100" nolu bellek adresine yazılmasıdır. Bu değerlerden hemen sonra ise bellekte A akümülatörünü '0' değerine ilk değer atamasını yapan "LDA# 0" komutu, ve bundan sonra da akümülatöre bellekte bulunan değerleri toplama işlemini yapan "ADD. 101" gibi "DIRECT" kipte belleğe erişen komutlar vardır.

3.2. BRA (koşulsuz atla) komutunun test edilmesi

BRA komutu programın normal akışını (yukardan aşağı) değiştirmek için kullanılmaktadır. Bu komutun doğru çalıştığını göstermek için basit bir akümülatöre değer yükleme komutunun üstünden atlayan bir program yazılacak ve doğru çalıştığı gösterilecektir.

```
0: ORG 100
100: 134 LDA# 1
102: 32 BRA+ 4
104: 134 LDA# 2
MEM[100-119]:134 1 32 4 134 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100:LDA#1 PC=102 A=1 B=0 X=0 C:0 Z:0 N:0
102:BRA+4 PC=106 A=1 B=0 X=0 C:0 Z:0 N:0
STOPPED AT:106
```

Şekil 3.3. BRA komutu ile koşulsuz komut atlama.

Şekil 3.3. teki örnekte görüldüğü üzere "LDA# 2" komutundan hemen önce bulunan "BRA+ 4" komutu ile bu komutun üstünden atlanmakta ve "A" akümülatörünün değeri değişmeden "1" olarak kalmaktadır. Böylece "BRA+ 4" komutunun doğru çalıştığı kesin olarak görülmüş olmaktadır.

3.3. BNE (eşit değilse atla) komutunun test edilmesi

Örnek olarak akümülatöre bellekte 100 adresinde kayıtlı değerin tek veya çift bir sayı olmasına göre indeks (IX) yazmacının değerinin artırılmasını sağlayan kodu yazalım. Bu kod aşağıda Şekil 3.4. te görülmektedir.

```
ORG 100
DB 2
DB 4
LDX# 0
LDA. 100
AND# 1
BNE+ 3
INX
LDA. 101
AND# 1
BNE+ 3
INX
END
LIST 100
EXEC 102
```

Şekil 3.4. Çift sayıları sayan program.

Bu programın çıktısı aşağıda Şekil 3.5. te görülmektedir. Görüldüğü üzere girdi olarak verilen “2” ve “4” sayıları çift olduğu için çift sayıların sayısını tutan “X” yazmacının değeri program sona erdiğinde 2 olmuştur.

```
MEM[100-119]:2 4 206 0 0 150 100 132 1 38 3 8 150 101 132 1 38 3 8 0
102:LDX#0 PC=105 A=0 B=0 X=0 C:0 Z:0 N:0
105:LDA.100 PC=107 A=2 B=0 X=0 C:0 Z:0 N:0
107:AND#1 PC=109 A=0 B=0 X=0 C:0 Z:1 N:0
109:BNE+3 PC=111 A=0 B=0 X=0 C:0 Z:1 N:0
111:INX PC=112 A=0 B=0 X=1 C:0 Z:1 N:0
112:LDA.101 PC=114 A=4 B=0 X=1 C:0 Z:1 N:0
114:AND#1 PC=116 A=0 B=0 X=1 C:0 Z:1 N:0
116:BNE+3 PC=118 A=0 B=0 X=1 C:0 Z:1 N:0
118:INX PC=119 A=0 B=0 X=2 C:0 Z:1 N:0
STOPPED AT:119
```

Şekil 3.5. Çift sayıları sayan program çıktısı, “2” ve “4” için.

Aynı program “1” ve “3” girdileriyle çalıştırıldığında ise sonuç aşağıdaki Şekil 3.6. daki gibi olur. Görüldüğü üzere “X” yazmacının değeri program sona erdiğinde sıfırdır. Buradan da “BNE+ 3” komutunun doğru bir şekilde görevini yerine getirdiği anlaşılmaktadır.

```
MEM[100-119]:1 3 206 0 0 150 100 132 1 38 3 8 150 101 132 1 38 3 8 0
102:LDX#0 PC=105 A=0 B=0 X=0 C:0 Z:0 N:0
105:LDA.100 PC=107 A=1 B=0 X=0 C:0 Z:0 N:0
107:AND#1 PC=109 A=1 B=0 X=0 C:0 Z:0 N:0
109:BNE+3 PC=112 A=1 B=0 X=0 C:0 Z:0 N:0
112:LDA.101 PC=114 A=3 B=0 X=0 C:0 Z:0 N:0
114:AND#1 PC=116 A=1 B=0 X=0 C:0 Z:0 N:0
116:BNE+3 PC=119 A=1 B=0 X=0 C:0 Z:0 N:0
STOPPED AT:119
```

Şekil 3.6. Çift sayı program çıktısı "1" ve "3" ile.

3.4. ASL (sola kaydır) komutunun test edilmesi

Bu komut eldeki sayıyı 2 ile kolayca çarpmak için kullanılabilir. Bu komutun çalışmasını doğrulamak için, Şekil 3.7. te kodu verilmiş olan program ile, bellekte 2 adres “4” ve “8” değerlerine eşitlenmekte ve her adresten bir sonraki bellek adresine bu değerlerin 2 ile çarpılmış değerini saklamak için ASL komutu kullanılarak hesaplanan değer STA komutu ile bellekte ayrılan adreslere yazılmaktadır.

```
ORG 100
DB 4
DB 0
DB 8
DB 0
LDA. 100
ASL
STA. 101
LDA. 102
ASL
STA. 103
END
LIST 100
```

Şekil 3.7. “4” ve “8” sayılarını iki ile çarpan program

Bu programın sonucu aşağıdaki Şekil 3.8. de verilmektedir.

```
MEM[100-119]:4 0 8 0 150 100 72 151 101 150 102 72 151 103 0 0 0 0 0 0
104:LDA.100 PC=106 A=4 B=0 X=0 C:0 Z:0 N:0
106:ASL PC=107 A=8 B=0 X=0 C:0 Z:0 N:0
107:STA.101 PC=109 A=8 B=0 X=0 C:0 Z:0 N:0
109:LDA.102 PC=111 A=8 B=0 X=0 C:0 Z:0 N:0
111:ASL PC=112 A=16 B=0 X=0 C:0 Z:0 N:0
112:STA.103 PC=114 A=16 B=0 X=0 C:0 Z:0 N:0
STOPPED AT:114
MEM[100-119]:4 8 8 16 150 100 72 151 101 150 102 72 151 103 0 0 0 0 0 0
```

Şekil 3.8. 4" ve "8" sayılarını iki ile çarpan programın sonucu

Program çıktısının en başında verilen ilk bellek listesinde ilk 4 bayt "4, 0, 8, 0" içermektedir. Programda "4" sayısı sola bir kere kaydırılarak "8" ve "8" sayısı sola bir kere kaydırılarak "16" elde edilmekte ve belleğe saklanmaktadır. Son bellek listesinde görülen "4, 8, 8, 16" değerleri istenildiği gibi "8" ve "16" değerlerinin ASL komutu tarafından hesaplandığını göstermektedir.

3.5. ASR (sağa kaydır) komutunun test edilmesi

Bu komut ile akümülatördeki sayı sağa bir bit kaydırılmakta ve ikiye bölünmüş bir değer elde edilmektedir. Bu komutun doğru çalıştığını göstermek için belleğe "10" ve "20" değerleri yerleştirilecek ve ikiye bölünerek "5" ve "10" değerleri elde edilmeye çalışılacaktır. Bu işlem için yazılmış olan kod Şekil. 3.9 da görülmektedir.

```
ORG 100
DB 10
DB 0
DB 20
DB 0
LDA. 100
ASR
STA. 101
LDA. 102
ASR
STA 103
```

Şekil 3.9. ASR komutu ile ikiye bölme.

Bu programın çalıştırılması ile elde edilen sonuçlar aşağıdaki Şekil. 3.10. verilmektedir. Görüldüğü üzere bellekte 100-103 adreslerinde bulunan “10 5 20 10” değerleri, doğru bir şekilde 10’un yarısının 5, 20’nin yarısının da 10 olarak hesaplandığını göstermektedir.

```
MEM[100-119]:10 0 20 0 150 100 71 151 101 150 102 71 151 103 0 0 0 0 0 0
104:LDA.100 PC=106 A=10 B=0 X=0 C:0 Z:0 N:0
106:ASR PC=107 A=5 B=0 X=0 C:0 Z:0 N:0
107:STA.101 PC=109 A=5 B=0 X=0 C:0 Z:0 N:0
109:LDA.102 PC=111 A=20 B=0 X=0 C:0 Z:0 N:0
111:ASR PC=112 A=10 B=0 X=0 C:0 Z:0 N:0
112:STA.103 PC=114 A=10 B=0 X=0 C:0 Z:0 N:0
STOPPED AT:114
MEM[100-119]:10 5 20 10 150 100 71 151 101 150 102 71 151 103 0 0 0 0 0 0
```

Şekil 3.10. ASR örnek programının ürettiği sonuçlar

3.6. A ve B yazmaçlarının birlikte kullanılması.

A ve B yazmaçlarını birlikte kullanan bir program örneği Şekil 3.11 de görülmektedir

```
ORG 100
LDA# 5
TAB
LDA# 23
ICB
INC
ABA
END
LIST 100
EXEC 100
```

Şekil 3.11. A ve B yazmaçlarını birlikte kullanan program.

Bu programın çalışma sonucu aşağıdaki Şekil 3.12. de görülmektedir. Önce A yazmacına 5 değeri yüklenip TAB komutu ile B yazmacına aktarılmakta, sonra da A yazmacına bu sefer 23 değeri yüklenmektedir. Bir sonraki adımda ise hem A hem de B yazmaclarının içindeki değerler INC komutları ile birer artırılmakta böylece yazmaçlarda 24 ve 6 değerleri elde edilmektedir. Bu iki değer ABA ile A yazmacında toplanmakta ve program sona ermektedir. Son değer olarak A yazmacında 30 sayısı elde edilmektedir.

```
0: ORG 100
100: 134 LDA# 5
102: 22 TAB
103: 134 LDA# 23
105: 92 ICB
106: 76 INC
107: 27 ABA
MEM[100-119]:134 5 22 134 23 92 76 27 0 0 0 0 0 0 0 0 0 0 0
100:LDA#5 PC=102 A=5 B=0 X=0 C:0 Z:0 N:0
102:TAB PC=103 A=5 B=5 X=0 C:0 Z:0 N:0
103:LDA#23 PC=105 A=23 B=5 X=0 C:0 Z:0 N:0
105:ICB PC=106 A=23 B=6 X=0 C:0 Z:0 N:0
106:INC PC=107 A=24 B=6 X=0 C:0 Z:0 N:0
107:ABA PC=108 A=30 B=6 X=0 C:0 Z:0 N:0
STOPPED AT:108
MEM[100-119]:134 5 22 134 23 92 76 27 0 0 0 0 0 0 0 0 0 0 0
```

Şekil 3.12. A ve B yazmaçlarını birlikte kullanan programın çıktısı.

SONUÇ

Bu tez de üniversitelerde verilen bilgisayar mimarisi derslerinde örnek olarak bir mikroişlemci tasarlanması ve bir assembler tarafından üretilen makine dili kodlarının bir emülatör aracılığı ile çalıştırılması ve makine düzeyi komutların yazmaçlar üzerindeki etkisinin doğrudan öğrenciler tarafından görülebilmesi amaçlanmıştır.

Bu tez kapsamında Motorola 6800 mikroişlemci temel alınarak, 8 bitlik bir mikro işlemci, bu işlemci için bir assembly dili ve assembly dilindeki komutları emüle edecek bir emülatör tasarlanmıştır.

Geliştirilen emülatör C++ dilinde kodlanmıştır. Bu sayede temel foksiyonlar C++ dilinde kodlandığı için yeni bir mikro işlemci mimarisi ve/veya makine dili tasarlanarak kolay bir şekilde emülatörün içine eklenebilecektir. Yapılan denemelerde LDA, STA, ADD, BRA, BNE, ASL, ASR gibi temel komutların doğru şekilde çalıştığı görülmüştür. Bu denemelerin daha karmaşık programlarla yapılması ve programların sorunsuz çalıştığının bütün komutlar için denetlenmesi zamana yayılarak yapılması gereken bir görevdir.

Öneriler ve Yapılacak İşler

Bu tez kapsamında yapılan işlere ek olarak şunlar da gerçekleştirilebilir. Metin etiketler program içinde tanımlanarak “LDA DEGISKEN1” veya “BRA+ LOOP” şeklinde komutlar girilebilecektir. Böylece bellek adreslemesinde olası hatalar engellenmiş olacaktır.

Yapılması gereken diğer bir işlem de yığıt veri yapısının oluşturularak JSR, PUSH; POP gibi komutların doğrudan emülatör tarafından desteklenmesinin sağlanmasıdır. Böylece

öğrencilere işletim sistemlerinde önemli yer tutan alt-yordam çalıştırma, parametre değeri geçirme gibi önemli işlevler de bu emülatör programı aracılığı ile gösterilebilir duruma gelecektir.

ÖZET

Bu tezin Birinci Bölümünde genel mikroişlemci mimarisi, Motorola 6800 mikroişlemci sinin mimarisi, emülatörler ve makine dili konusunda gerekli bilgiler verilmektedir.

İkinci Bölüm de, örnek bir 8-bit işlemci makine dili tasarlanmakta, ayrıntıları sunulmakta ve bu makine dilinin komutlarını emüle eden bir C++ yazılımının yapısı açıklanmaktadır.

Üçüncü Bölüm de, geliştirilen bu makine dili ve emülatör ile yapılmış olan örnek assembler kod denemeleri anlatılmakta, bu kodların çalışması ve sonuçları açıklanmaktadır.

Sonuç bölümde ise geliştirilen bu makine dili ve emülatör'den edinilen deneyim anlatılmakta ve eğitim amaçlı olarak bu makine dili ve emülatör yazılımının nasıl kullanılacağı ve daha da geliştirilmesi için nelerin yapılabileceği tartışılmaktadır.

Bilgisayar mimarisi derslerini öğrenirken gerçek bir mikroişlemcinin mimarisini, o işlemcinin makine dili komutlarını ve o işlemciyi çevirme dilinde(assembly language) programlayarak öğrenmek önemlidir.

Öğrenciler, makine dili modellemesi ile, bir donanımı kurup onun emülasyonunu gerçekleştirdikleri zaman daha iyi öğrenirler. Multimedia Logic (MML) grafiksel bilgisayar mimarisi, uygun G/Ç desteği sağlayarak bilgisayar tasarımı ve emülasyonuna olanak sağlayan yazılımlardan biridir. Ancak MML kullanılarak yapılan emülasyon tasarımları, MML ye bağımlı kalmaktadır.

Bu çalışmada 8-bitli basit bir mikro işlemci emülatörü ve bu emülatörün assembly dili tasarımı yapılmıştır. Çalışmada yaygın olarak kullanılmış başarılı bir ürün olması, basitliği ve kolay anlaşılır olması nedeniyle Motorola 6800 işlemcisini komutları örnek alınmıştır. Emülatör ve assembly dili C++ dili kullanılarak kodlanmıştır. MML kullanılmadığı için de MML den bağımsız bir emülatör ve assembly dili olmuştur.

En sık kullanılan makine dili komutları, akümülatör yazmacına değer yükleyen ve akümülatördeki değerleri işleyen (toplama, belleğe saklama gibi) komutlardır. Bu tezde tasarlanan işlemcinin bir adet 8-bit akümülatörü ("A"), bir adet 16 bit indeks yazmacı (IX), ve 16-bit program sayacı (PC) vardır.

Akümülatör komutları:

LDA <değer> – "A" akümülatörüne bir değer yükle
STA <bellek adresi> - "A" akümülatörünün değerini belleğe kopyala
ADA <değer> veya <bellek adresi>

Diğer önemli komut kümesi program akışını dallandırma ("branch") komutlarıdır. Bu komutlarda "göreceli" adresleme kullanmakta olup "-128" ile "+127" arasında bir

değeri program sayacı (PC) yazmacına ekleyerek program kontrolünü elde edilen bu yeni adresteki komutu bir sonraki komut olarak çalıştıracak şekilde değiştirmek mümkündür. Böylece değişik koşullar altında programın değişik kısımlarının çalıştırılması ve farklı işlem yapılması gerçekleştirilmektedir.

Bu komutların yeterli işlevselliği sağlayacak küçük bir alt kümesi bu çalışmada gerçekleştirilen emülatör tarafından tanınmakta ve emüle edilmektedir, Bu komutlar aşağıda verilmiştir.

Dallanma komutları:

BRZ	Eğer "sıfır" ise
BEQ	Eğer "eşit" ise
BLT	Eğer "küçüktür" sıfır ise
BGT	Eğer "büyüktür" sıfır ise
BNE	Eğer "eşit değil" ise

Aritmetik ve mantık kaydırma komutları akümülatörün içindeki değeri sola veya kaydırma ve döndürme işlemlerinin gerçekleştirmek için kullanılırlar. Bu çalışmadaki emülatör aşağıdaki kaydırma ve döndürme komutlarını gerçekleştirebilmektedir.

Kaydırma ve döndürme komutları:

ASL	Aritmetik sola kaydır
ASR	Aritmetik sağa kaydır
LSR	Mantıksal sağa kaydır
ROR	Sağa döndür
ROL	Sola döndür

Diğer bir önemli komut kümesi de STATUS (durum) yazmacında bulunan C (carry/elde), Z (zero/sıfır), N (negatif) gibi bayrak değerlerini değiştiren komutlardır.

SEC	C = 1
CLC	C = 0
SEZ	Z = 1
CLZ	Z = 0
SEN	N = 1
CLN	N = 0

İşleyiş ve veri tanımlama komutları:

"ORG" : programın bellekte başlangıç adresini vermektedir.

"DB" DEFINE BYTE: Bellekte önceden tanımlanmış değerlerin yerleştirilmesine izin vermektedir.

“END” : Programın çalışmasını sona erdirmektedir.

Emülatörü test etmek için bazı deneme uygulamaları yapılmış ve LDA, STA, ADD, BRA, BNE, ASL, ASR gibi temel komutların doğru şekilde çalıştığı görülmüştür.

Geliştirilen emülatör C++ dilinde kodlanmıştır. Bu sayede temel foksiyonlar C++ dilinde kodlandığı için yeni bir mikro işlemci mimarisi ve/veya makine dili tasarlanarak kolay bir şekilde emülatörün içine eklenebilecektir.

МАЗМУНУ

Бул проектин биринчи бөлүмүндө жалпы микропроцессор архитектурасы, Motorola 6800 микропроцессорунун архитектурасы, эмулятор жана машина тили темасындагы керектүү информациялар берилген.

Экинчи бөлүмдө мисал үчүн бир 8-биттүү микропроцессор машина тили проектирленди. C++ тилинде жасалган машина тилинин командаларын эмуляциялоо жеткиликтүү ачылып жазылды.

Үчүнчү бөлүмдө жогоруда көрсөтүлгөн машина тили жана эмулятор аркылуу жасалган мисалдардын ассемблердеги коддору түшүндүрүлгөн. Ал коддордун иштеши жана жыйынтыктары көрсөтүлгөн.

Аяккы бөлүмдө көрсөтүлгөн машина тили жана эмулятордон алынган тажрыйба түшүндүрүлдү. Ушу машина тилин окуп үйрөтүү үчүн кантип колдонула турганы жана эмулятор менен машина тилин дагы өркүндөтүү үчүн кандай мүмкүнчүлүктөрү бар экендиги жазылган.

“Компьютердин архитектурасын” окуу курсунда реалдык микропроцессорду окуп түшүнүү үчүн – микропроцессордун түзүлүшүн, машиналык командаларын – микропроцессордун ассемблер тилин окуп үйрөнүү өтө маанилүү.

Студенттер бул айтылган сабакты окуп үйрөнүүдө процессорду курганды жана аны техникалык камсыздыгын эмуляция кылганды, машина тилин моделдештирип имитация кылып үйрөнсө сабакты жакшы өздөштүрүшөт. Мисалы бул Multimedia Logic (MML) болушу мүмкүн, ал кандайдыр бир графикада компьютердин архитектурасы болот жана анда кириш-чыгыш мүмкүнчүлүктөрү бар. Бирок MML колдонуп жасалган эмуляция проектери MMLге көз каранды болуп калышат.

Бул жумушта 8 биттик микропроцессорду эмуляциялоо программасы каралат, анда микропроцессордун түзүлүшү жана аны менен байланышкан ассемблер тилинин иштөөсү каралат. Motorola 6800 микропроцессорунун архитектурасы жана ага туура келген машина тили мисал катары көрсөтүлгөн. Ал жөнөкойлүгү менен окуп үйрөнүүгө жакшы.

Машина тилинде өтө көп колдонуучу командалардын бири болуп кандайдыр бир маанини машинага жүктөо жана аны менен болгон амалдар-командалар эсептелет. Биз түзгөн микропроцессор 8 биттик аккумулятору(A) болот, анын 16 биттик индекстөөчү регистри бар (IX) жана 16 биттик командаларды эсептөөчү регистрден турат (PC).

Accumulator командалары

LDA <маани>: “A” аккумуляторунун регистрине маанини жүктөө. Бул командanын параметринин мааниси кезектеги аткарылуучу командanын номери же эстин адреси болушу мүмкүн(түз, кеңейтилген же индекстүү түрдө).
STA <эстин адреси>: “A” аккумуляторуна эстин адресин жүктөйт. Адрес төмөнкү түрлөрдүн бири болушу мүмкүн - direct mode (1 байт, башкача айтканда 0..255 чейинки бүтун сан), extended mode же indexed mode.
ADA < маани >: “A” аккумуляторуна маанини кошот.

Тармактануу командалары

Негизги командалардын көптүгүнө тармактануу командалары кирет жана алар программанын аткарылуу ирээтин өзгөртүп туруу үчүн колдонулат. Тармактануу командаларында “салыштырмалуу адресс” колдонулат. Алардын мааниси -128 тен +127 ке чейин болушу мүмкүн. Ал регмстрге жүктөлөт жана кийинки аткарылуучу командanын адресин көрсөтөт.

Алардын кээ бир түрлөрү бул эмулятордун функционалдуулугуна зарыл болгондору бул иште колдонулган. Алардын түрлөру:

BRZ	тармактан эгерде ноль
BEQ	тармактан эгерде барабар
BLT	тармактан эгерде кичине
BGT	тармактан эгерде чоң
BNE	тармактан эгерде барабар эмес

Арифметикалык жана логикалык командалар регистрдин маанисин солго же оңго жылдыруу жана өзгөртүү үчүн колдонулат.

Жылдыруу жана ротация командалары

ASL	солго арифметикалык жылдыруу
ASR	оңго арифметикалык жылдыруу
LSR	оңго логикалык жылдыруу
ROR	оңго ротация
ROL	солго ротация

Регистрдин желекчесин STATUSун өзгөртүү үчун төмөку командалар көптүгү колдонулат: C(өзгөртүү), Z(нөл) и N(оң эмес) желекчклери.

SEC	C = 1
CLC	C = 0
SEN	N = 1
CLN	N = 0
SEZ	Z = 1

CLZ Z = 0

Программаны башкаруу командалары жана программада колдонулуучу маанилерди аныктоочу командалар дагы бул микропроцессорду түзүүдө колдонулган, ошондой эле алардын кодун машина тилине генерацялап эске сактоо каралган:

ORG(origin): бул команда аркылуу ассемблер колдонуучу эстин областы көрсөтулөт.

DB(АНЫКТАМА BYTE): прогамманы башталган чекитин көрсөтүү үчүн колдонулат.

END: ассемблер программасынын аягын көрсөтөт.

Эмулятору текшерүү (тестирлөө) үчүн жасалган жана бул проекте колдонулган ассемблер тилиндеги төмөнкү командалар кирет: LDA, STA, ADD, BRA, BNE, ASL жана ASR.

Эмулятор C++ тилинде жазалган. Бул C++ тилин колдонуу эмуляторду өзгөртүүгө же жаңы микропроцессорлорду проектирлөөдө жана ошондой эле жаңы машина тилиндеги командаларды кошуп толуктоодо абдан ыңгайлуу.

KAYNAKLAR

[1] Brorsson, M. "MipsIt—A Simulation and Development Environment Using Animation for Computer Architecture Education", Proceedings of the 2002 workshop on Computer architecture education, Anchorage, Alaska, USA, 2002

[2] Sugawara, Y. ve Hiraki, K. "A computer architecture education curriculum through the design and implementation of original processors using FPGAs," Proceedings of the 2004 workshop on Computer architecture education, Munich, Germany, 2004

[3] Becvar, M., Pluhacek, A. ve Danecek, J. "DOP: a CPU core for teaching basics of computer architecture," Proceedings of the 2003 workshop on Computer architecture education, San Diego, CA, USA, 2003

[4] Stanley, T.D. ve Wang, M. "An Emulated Computer with Assembler for Teaching Undergraduate Computer Architecture," Proceedings of the 2005 workshop on Computer architecture education, Madison, Wisconsin, USA, 2005

[5] Stanley, T.D., Xuan, T.Q., Fife, L. ve Colton, D. "Simple eight bit, emulated computers for illustrating computer architecture concepts and providing a starting point for student designs," Proceedings of the ninth Australasian conference on Computing education, Ballarat, Victoria, Australia, 2007

[6] Uduguma, L.S.K. ve Geeganage, J.C. "Students' experimental processor: a processor integrated with different types of architectures for educational purposes", Proceedings of the 2006 workshop on Computer architecture education, Boston, Massachusetts, USA, 2006

[7] Motorola "M6800 Microprocessor Applications Manual", Phoenix, Arizona, USA, 1975

[8] Softronics Inc., "Multi Media Logic", http://www.softronix.com/logic.html , 1.4 release, January 22, 2004

[9] Bartee, T.C. Computer Architecture and Logic Design. McGraw Hill, New York, USA, 1991

ÖZGEÇMİŞ

Doğum yeri ve yılı :

Öğr. Gördüğü kurumlar	Başlama yılı	Bitirme yılı	Kurum adı
Lise	1973	1976	Ankara Gazi Lisesi
Lisans	1976	1982	O.D.T.Ü. Elektronik Müh. Böl.
Yüksek lisans	2007	2010	Kirgizistan-Türkiye Manas Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Ana Bilim Dalı

Medeni durumu : Evli
Bildiği yabancı diller ve düzeyi : İngilizce, Çok iyi

	Çalıştığı kurumlar	Başlama ve ayrılma tarihleri	Görev
1	Kirgizistan-Türkiye Manas Üniversitesi	11/2001 -07/2010	Bilgisayar Mühendisliği Bölümü Öğretim Görevlisi
2	Ahmet Yesevi Üniversitesi Türkistan Şehri Kazakistan	11/1995 -8/1999	Bilgisayarlı Test Merkezi Yöneticisi(11/2000-8/2001 Bilgi İşlem daire Başkan Yrd.)
3	Setkom A.Ş Ankara, Türkiye	1/1995 -6/1995	Elektronik Atölye Şefi
4	3.Hava İkmal Bakım Merkezi Ankara, Türkiye	11/1988 -7/1994	Proje ve Test Mühendisi
5	E.C.S. A.Ş. Ankara, Türkiye	1/1987 -8/1988	Elektronik Atölye Şefi

	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010
	BİLGİSAYAR MİMARİSİ ÖĞRETİMİ İÇİN BASİT 8-BİT İŞLEMCİ EMÜLASYONU ÜZERİNE BİR ÇALIŞMA (YÜKSEK LİSANS TEZİ)	Mehmet Selim ELMALI	BİŞKEK 2010