Subject: i

۱۳۹۷/۰۱/۲۸

جلسه یازدهم:

www.googoolia.com

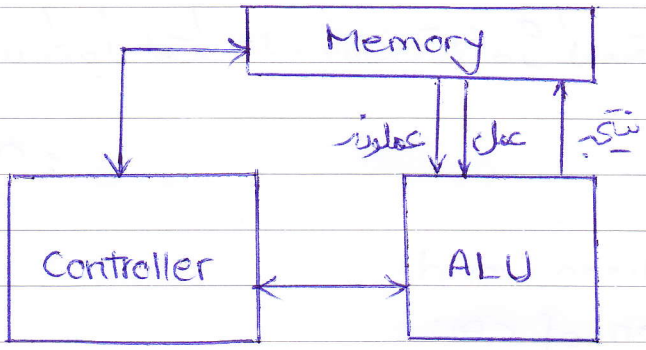
mamsadegh@googoolia.com

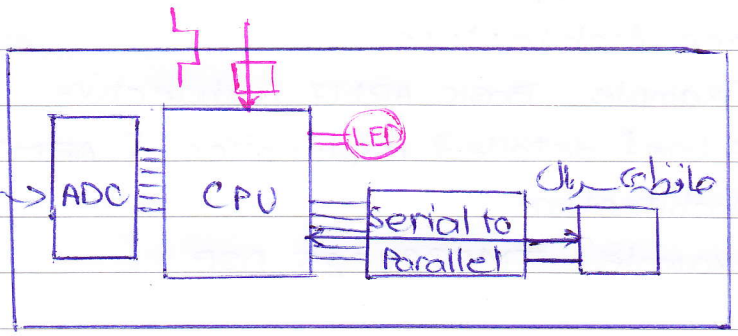
AVR ?

FPGA ?

CPU: Central Processing Unit

معماری:

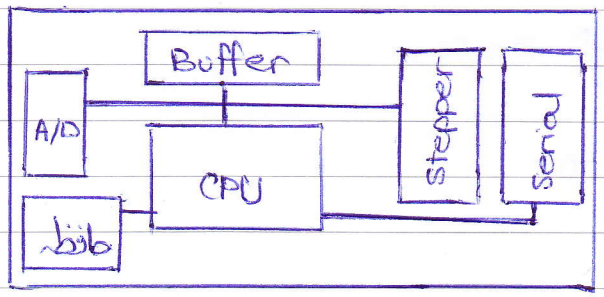




Stepper Motor

Peripheral : A module or unit which is responsible for communication between CPU and the environment

Microcontroller :



Subject: 2

۸۹ | 7 | 30

تکلیف: CPU موبایل ، فرکانس کاری

AVR: هسته پردازشی از شرکت Atmel

Assembly
C

PIC18 :

PHY : Physical Interface

بین PIC و شبکه

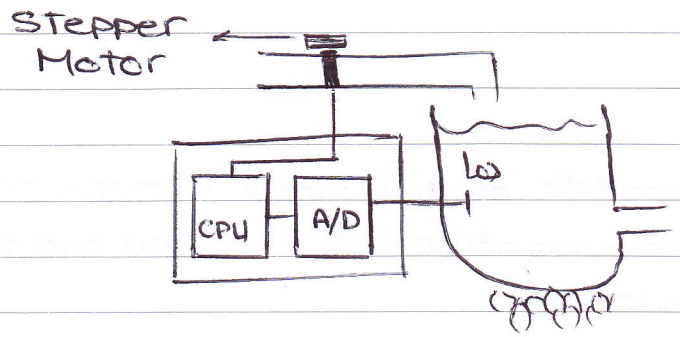
PIC یک web-surfer کوچک است که می‌تواند به شبکه متصل شود.

مقدمه‌ای بر CPU :

تعریف:

- A tour of the silicon world!
- Basic definition of CPU
- Von Neumann Architecture
  - Example : Basic ARM7 Architecture
  - A brief detailed explanation of ARM7 Architecture
- Hardvard Architecture
  - Example: TMS320C25 DSP

مثال : CPU



History of CPUs:

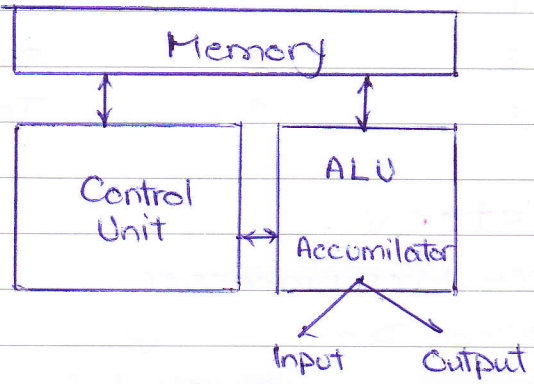
- 4004
- TMS100
-

Subject:3

Von Neumann Architecture

Same Memory
- Program
- Data

Single Bus



ARM7 T CPU

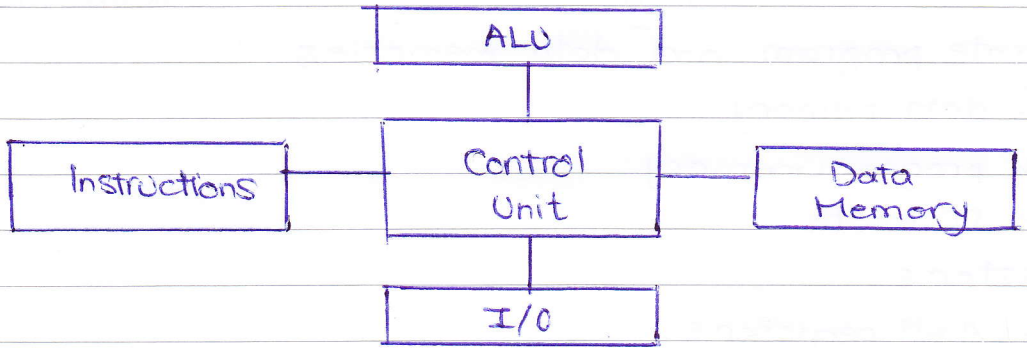
Bus — تنها یک باس ارتباط با محیط → مزیت: حجم و جریان کمتر / عیب

Multiplier 32×8

ALU +/-/ اعمال منطقی

Register Bank : حافظه‌ی کوچک : ALU ← باس ← باس ← عملوند

Hardvard Architecture:



TMS320C25 DSP:

Digital Signal Processor (DSP)
- Data Bus (انتقال عملوندها)
- Program Bus (انتقال دستور) ← مجزا

Silicon Market:

Rank 2009	Rank 2008	Company
8	← 12	AMD

Subject: 4

Rank 2009	Rank 2008	Company	Country
1	1	Intel	USA
2	2	Samsung	S Korea
3	3	Toshiba	

History :

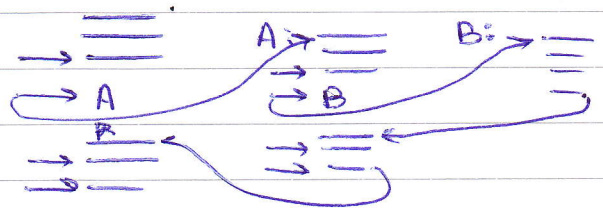
- 4004 Microprocessor
  - ~ Silicon Die : بستر
  - ~ Package : Physical Appearance : Dual InLine Package (DIP)
  - ~ 1971
    - Just as a calculator
    - 4 bit CPU
    - ALU : 4 bits
    - Instructions : 8 bits

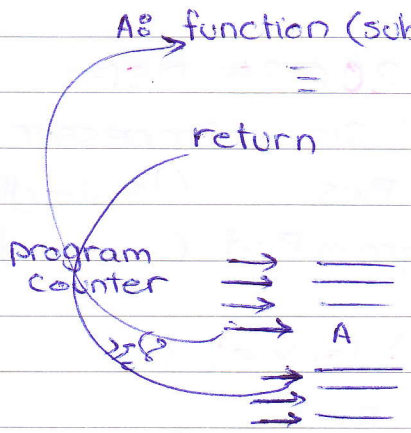
(CPU : Core_i7 ; 32 bit instructions ; 64 bit calculations - ALU)

(CPU : Itanium 64 bit ; 64 bit instructions ; 64 bit calculations)

معمولاً از روی ALU می‌توان تعداد بیت CPU را تشخیص داد.

  - ~ Separate program and data memories
    - 1 K data memory
    - 4 K program memory
      - PC width
  - ~ Registers
    - 16 x 4 bit registers
  - ~ Stack
    - 4 Level stack
  - ~ Instructions
    - 46





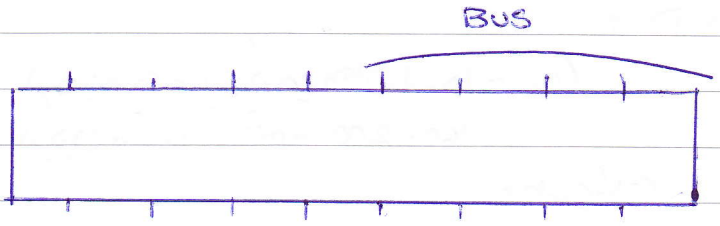
Program Counter
نشان‌دهنده آدرس دستورالعمل بعدی که باید اجرا شود

Nokia 7610 : TI OMAP1510 123 MHz Processor

Subject: 5

| 89 | 7 | 4 |

4004 :



باس 16 بایتی

Supported Chips :

4001 : 256 bytes ROM

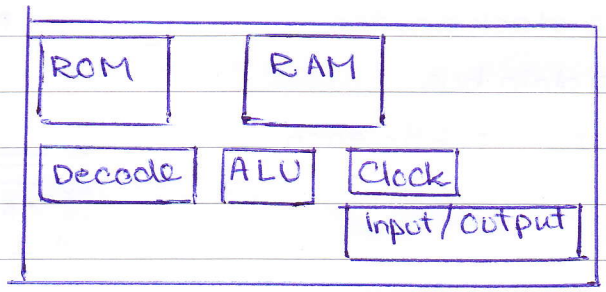
4002 : 40 bytes RAM

⋮

Texas Instruments TMS 1000

Silicon Die

اولین میکروپروسسور



ColdFire :

هسته‌ی پردازشی 32bit

Freescale

Motorolla (قسمتی از این شرکت از آن جدا شده و شرکت Freescale شد)
spinoff

می‌توان به عنوان web-surfer به کار رود.

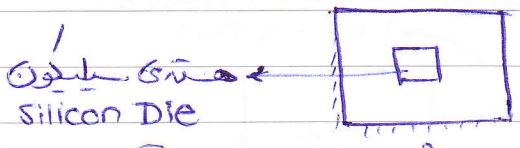
Pentium 3 :

پردازنده

1 GHz : فرکانس کاری

Package : Pin grid array

پایه‌ها از زیر هسته بیرون



مساحت آن متناسب با تعداد ترانزیستورها = 80 mm^2

Subject: 6

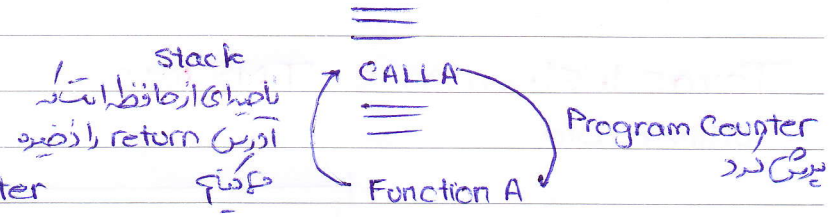
Pentium D :

تعداد پایه‌ها خیلی بیشتر شده. (توی‌های خیلی کوچیکن پایه‌ها هستن)
ابعاد کوچیک شده و مساحت سیلیکون بیشتر = $160 - 200\ mm^2$

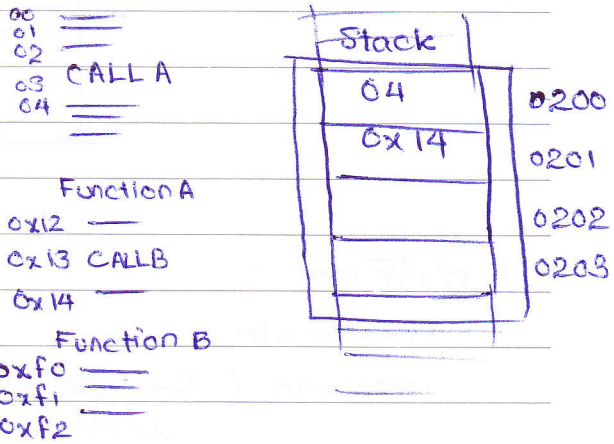
3 GHz : فرکانس کاری

Intel 8080 :

- 1974
- 8 bit CPU
- Buses
  - 16 bit address bus
    - what range?
    - 16 bits PC
    - 16 bits stack pointer
  - 8 bits data bus



- 7 registers : 8 bits each



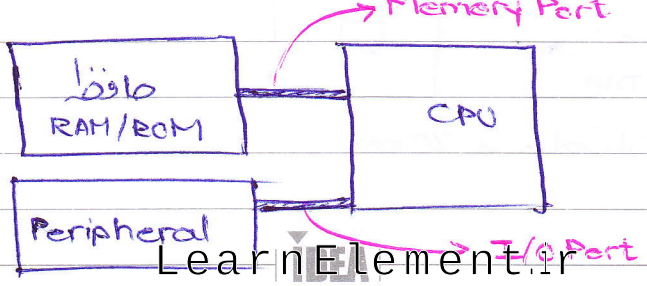
Register : Stack Pointer

ابتدا 0200

CALLA 0200

CALLB 0201

- Some registers could join to make 16 bits registers. (برای address)
- Separate
  - Memory Port
    - To talk to external memory
  - I/O Port
    - To talk to external peripherals and devices



Subject: 7

89 / 7 / 6

Zilog Z80:

- 1976
- An improved 8080
  - ~ 80 additional instructions
    - Block move instructions — یک تعدادی دستور را همزمان انتقال دهیم
    - Bit manipulation — در رجیسترها به تک تک بیت‌ها دسترسی داشته باشیم
  - ~ 2 register banks
    - Suitable for interrupt handling
- Federico Faggin
  - ~ Left Intel at 1974
  - ~ After his work on 8080
  - ~ Founded Zilog
- Clock Frequency:

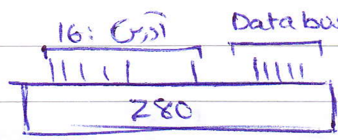
  2.5 MHz to 20 MHz (NMOS to CMOS)
- Memory Interface:

  Z80 capable generating DRAM refresh signal itself

حافظه‌های فرار (برق قطع می‌شود فرار می‌کنند)
→ Static (1 یا 2 Mbit = $20) — صرف‌نظر از هزینه بالا، حجم خیلی اندک است.
→ Dynamic (16 Gbit byte = $20) — حجم خیلی بیشتر می‌توانند باشند اما احتیاج به Refresh کردن دارد و RAM: و عموماً سریع‌تر هستند

- CP/M

  compatible with 8080
- Etensions to Z80:



- Two Register Banks

حافظه‌ها:

دینامیک: خیلی دیدیم

Synchronous Dynamic RAM : SDRAM

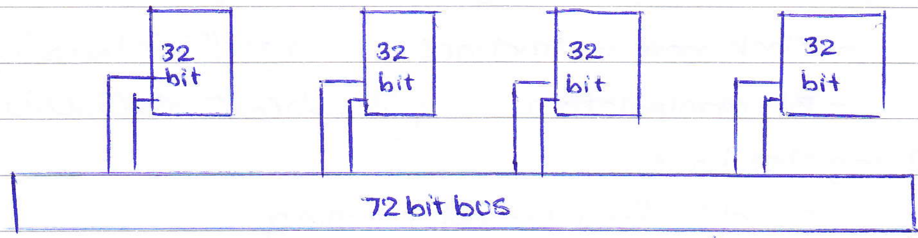
با پالس ساعت هماهنگ می‌شود.

Synchronous Static RAM : SSRAM

اگر بخواهند سرعت را بالا ببرند از [illegible] استفاده می‌کنند

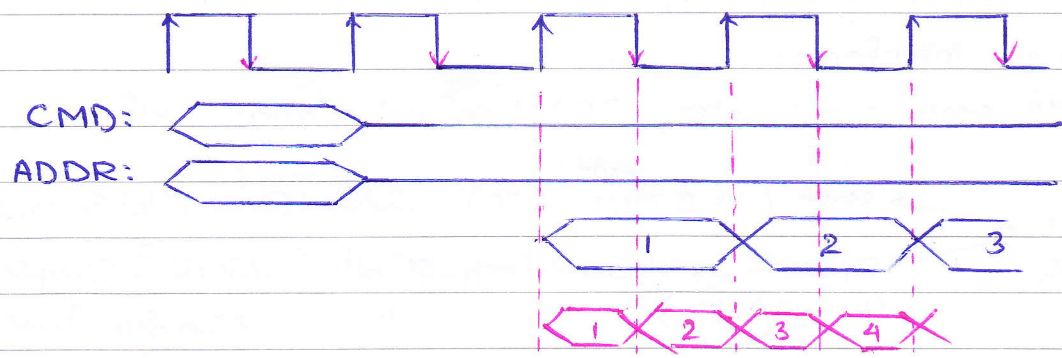
= SDRAM
حجم حافظه

- 256 Mbytes = 256 x 8Mbit = 2048 Mbit   ↕ اختلاف
32 M x 72 = 2304 Mbit



ECC   72 bit → 64 bit = داده
72 bit → 8 bit = parity ~> performance کمتر است

چون module های حافظه این طور نیستند و در خانه از این ها استفاده نمی‌کنیم.



Dual	Quad	SDRAM
Data	Data	DDR SDRAM
Rate	Rate	

- DDR 400 سرعت حافظه را تعیین می‌کند Dual ⇒ 200 MHz : Clock
64 x 8 : Configuration

- DDR 2 - 667 → Version 2   تأخیرها کم می‌شود
(DD2 - 667 ⇒ 333.333 MHz) اگر

0x1000   مثال →   MOV A , 1000h   DRAM با اعمال روبه‌رو مشکل
0x1010   MOV 1010h , A   دارد و تأخیر دارد (زمان RA:

(Random Access Time = t_{RC}

و مشابه DRAM و SRAM زیرا در SRAM تأخیر است ←

Subject: 9

- DDR3
  ↳ Version 3

SRAM :

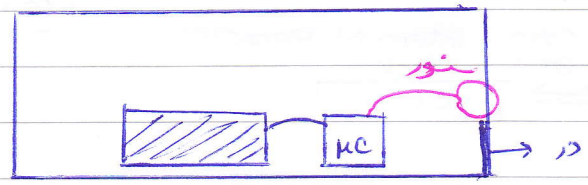
CY7C1462
SRAM
Cypress
36 Mbit ≡ $70
18 bit
133 MHz

خواندن و نوشتن روی این حافظه آسان است

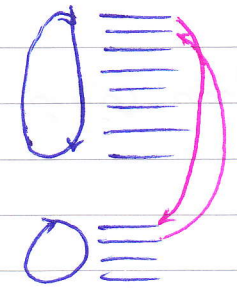
کاری که DRAM در 60ns انجام می‌دهد SRAM در 21ns انجام می‌دهد

ادامه : Z80 :

ساختار داخلی یک تفاوت عمده با ساختار داخلی بقیه CPUها دارد و آن این است که دو Register Bank دارد که هر کدام رجیسترها 8 بیت هستند (البته برای آدرس دهی دو رجیستر می‌توانند بچسبند به هم). این ویژگی برای مواردی که وقفه وجود دارد به کار می‌آید.

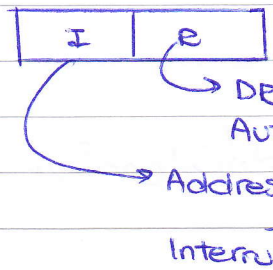


با بازشدن در برنامه متوقف شود
و با بسته شدن در برنامه ادامه داشته باشد

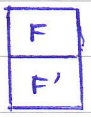


یا می‌خواهیم هر جای برنامه که هستیم با بازشدن در برنامه متوقف شود و پرش کند به طبقه دوم که چک می‌کند در بسته شد یا نه.

یک راه برای وقفه این است که حین وقفه از register bank دوم استفاده کنیم و برای ادامه‌ی برنامه به register bank اولی برمی‌گردد.



رجیستری که flagها در آن نگه‌داری می‌شود



Subject: IO

89 | 7 | 11

Registers (Z80):

A: Accumulator

F: Flag

SP: StackPointer: 16 bit

PC: Program Counter: 16 bit

Assembly Program:

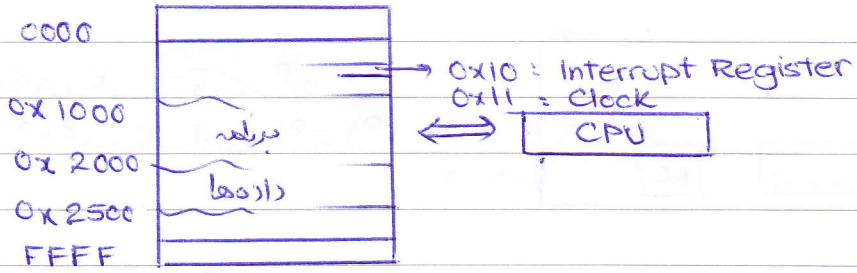
```
ORG   100 H
LD    HL, 1234 H
LD    A, (HL)
INC   HL
ADD   A, (HL)
```

تمام انتقال داده‌ها با استفاده از LD

حجم رام: 512 Mbyte $\Rightarrow \frac{2^9 \times 2^{20} \times 9 \text{ bit}}{2^3} = 576$ Mbytes

با بیت Parity

سیستم آدرس‌دهی:

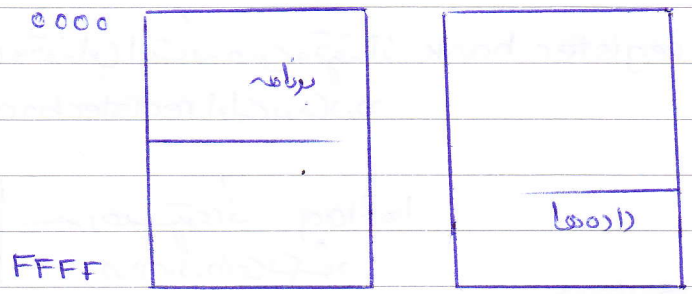


Memery Map

چه بخشی از حافظه مربوط به چه بخشی است

معماری Harvard: Instruction Bus ↗ جدا
Data BUS

آدرس‌ها پایان است



بنابه دستور مشخص است کدام حافظه (داده یا برنامه) مورد استفاده است

```
MOV A, 10 h
MOVC A, 10h
```

Subject: 11

A: [ ]

* LD A, 56h : Immediate Addressing

مقدار 56h در رجیستر ریخته می‌شود (خود دستور حاوی مقداری است که قرار است در رجیستر ریخته شود)

حافظه:

56h	0x2001

* LD A, (2001h) : Direct Addressing

رجیستر: A: [ ] HL: [0x2001]

* LD A, (HL) : Indirect Addressing

56h	0x2001

LD A, H — A, H هردو رجیستر هستند (مقدار H درون A ریخته می‌شود)

LD A, (H) — محتوای خانه‌ای از حافظه که این خانه در رجیستر وجود دارد درون A ریخته می‌شود (H: آدرس)

ادامه Z80:

Addressing Modes

- Immediate
- Page Zero (used for jumps and calls)
- Relative (used for jumps and calls)
- Extended Addressing (used for jumps and calls)
- Indexed Addressing

LD A, (IX+1)

LD A, (IX-1)

خانه‌های حافظه:

1	0x1000
2	0x1001
3	
4	

LD IX, 1000

LD A, (IX+1) — مقدار خانه‌ی 1001 را می‌ریزد در A

LD A, (IX-1)

مثل شبیه Indirect Addressing

Subject: 12

```
ORG  100H
LD   HL, 1234H
LD   A , (HL)
INC  HL
ADD  A, (HL)
INC  HL
LD   (HL), A
END
```

برنامه کجای حافظه قرار گیرد و دستورات

Address of first number

AMD AM2901 :

- 4 bits CPU

  4 bit slice processor راحت تا هم قرار گیرند برای انجام محاسبات سنگین تر

- Contained ALU and control signals

  8bit ALU consists of two 4 bit ALU

- Contained ALU
- محاسبات انجام می‌دهد

Intel MCS-51 (8051) :

- 1977
- Microcontroller

  On chip RAM and ROM حافظه روی خود چیپ وجود دارد

- 2 byte instruction set
- Extension by Siemens and Texas Instruments
- Four separate register sets چهار دسته رجیستر (در Z80 دو رجیستر بانک بود)
- internal static RAM: 128 bytes

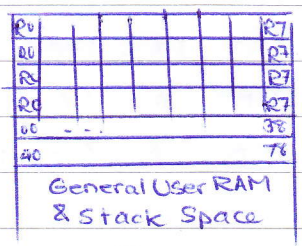
رجیسترها بخشی از RAM هستند

این دو دستور یک طور را انجام می‌دهند.

```
MOV R1, #2
MOV 01h, #2
```

طاق آدرس حافظه

Memory Map



Register Bank 0
Register Bank 1
Register Bank 2
Register Bank 3
Bits 00-3F
Bits 40-7F

Subject: 13

Memory Map :

0xFF	Upper 128 RAM Indirect Addressing	Special Function Registers
0x80		
0x7F	Direct and Indirect Addressing	
0x30		
0x2F	Bit Addressable → بیت‌ها قابل دسترسی‌اند	
0x20		
0x1F	General Purpose Registers	
0x00		

```
MOV   R0 , #80        Indirect Addressing
MOV   @R0 , #0F
MOV   80h , #0F       Direct Addressing
```

Instruction Sample :

```
SETB
```

- Supported peripherals
  - UART (رابطی برای داده‌ها را سریال بکند) serial Interface
  - Timers
  - I2C
  - SPI (interface فیزیکی)
  - USB ⟶ peripheral ⟵ host
  - CAN
  - PWM generator
  - Analog comparator
  - A/D and D/A
  - RTC
    - ~A computer clock
    - ~ keeps accurate time and date
    - ~ keeps accurate time

Subject: 14

| 89 | 7 | 13 |

Z80: LD A, 77h 2byte ⇒ CISC

LD A, (1000h) 3byte

طول دستورها فرق می‌کند

RISC: طول دستور ثابت است و تعداد دستورها کم

CISC: طول دستور متغیر و تعداد دستورها زیاد است

RISC: دستورهای ریاضی هیچ‌گاه نمی‌رود سراغ حافظه

ADD A, (HL)

CISC: دستور روبه‌رو وجود دارد

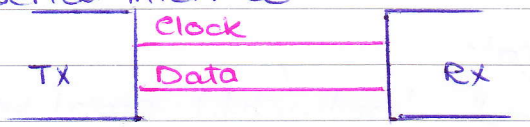
TMS320C 6201 : نمونه از پردازنده‌ی RISC

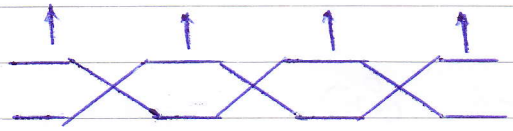
8051 ادامه:

UART:

Universal Asynchronous receive transmitter

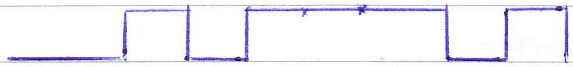
Synchronous Serial Interface:





هر چیزی که دوست داریم می‌توانیم خروجی Clock را بالا ببریم

Asynchronous



گیرنده و فرستنده سرعت انتقال داده اینبار توافق می‌کنند. (Baud Rate)

Baud Rate: 9600 bit/s

سرعت انتقال را خیلی بالا نمی‌توانیم ببریم و بزرگترین آن

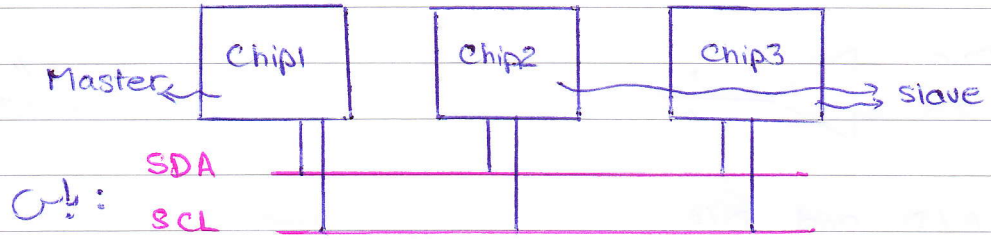
Baud Rate: 115200 bit/s

است.

USB : انتقال داده با سرعت بالا بدون انتقال Clock (Clock در واقع روی خود داده سوار است)

Timer : مقدار دوره را میتوان با آن اندازه گرفت.

I2C : استاندارد انتقال سریال

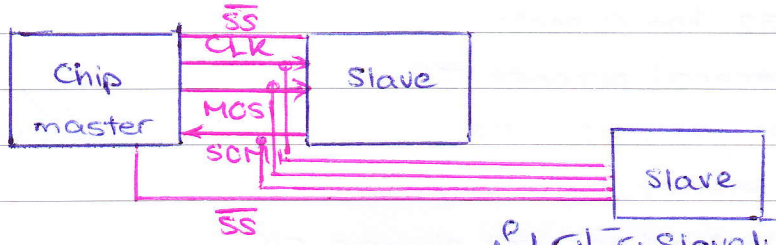


باس I2C پروتکل دارد. هم send و هم recieve این داده روی همین دو پایه صورت میگیرد.
I2C : phillips، بیشتر برای چیپ های ویدیویی

SPI :

Serial peripheral interconnect

روش انتقال داده سریال



یک Master و چندتا Slave میتوان باشد

MOSI : Master Output / Slave Input

SOMI : Slave Output / Master Input

SS : Slave Select

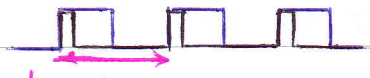
اگر SS باشد : active low است یعنی وقتی صفر است فعال است

CAN :

یک باس سریال است در صنعت استفاده میشود. موضوع تحقیق

Controller area network

PWM :



فرکانس همیشه ثابت است

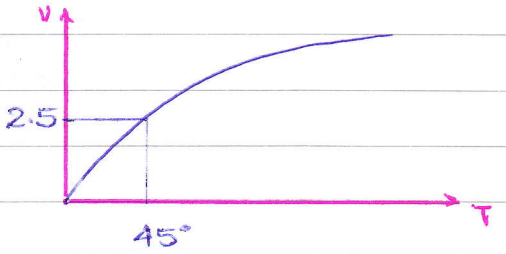
اطلاعات در پهنای پالس ارسال میشوند

Subject: 16

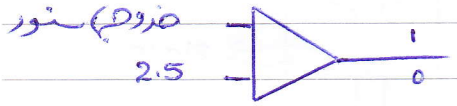
۸۹ | ۲ | ۱۶

- Analog Comparator :

سنسور دما :



مقایسه کننده‌ی آنالوگ :



- A/D and D/A
- RTC

Real Time Clock

برای اینکه وقتی برق رفت تاریخ به هم نخورد.
برای اینکه میلرو دقت خوبی برای اندازه‌گیری زمان را ندارد
چیپ است و باتری دارد

Special Function Registers

8051/52 has 4 ports

- General purpose I/O
- P0, P1, P2, P3

پایه‌های چیپ را کنترل می‌کند

Stack Pointer:

- Instructions that change SP

CALL, RET, PUSH, POSH

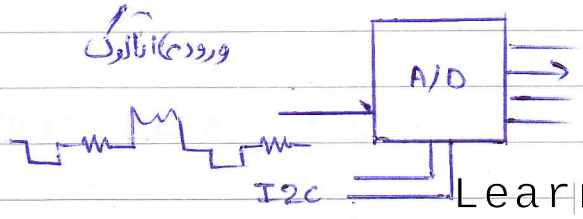
PCA 82 C252
ADV7202 → A to D ویدیو
AT 25 DF021
TCM 3105

سرعت بالا

سرعت پایین

خودرو : CAN سیستم‌های صنعتی و خودرو
تلویزیون : I2C
حافظه‌ی گوشی موبایل : SPI
مودم : UART

ADV7207



SP points to the tail of the stack (Special Purpose Register)

PUSH A:

SP ← SP + 1

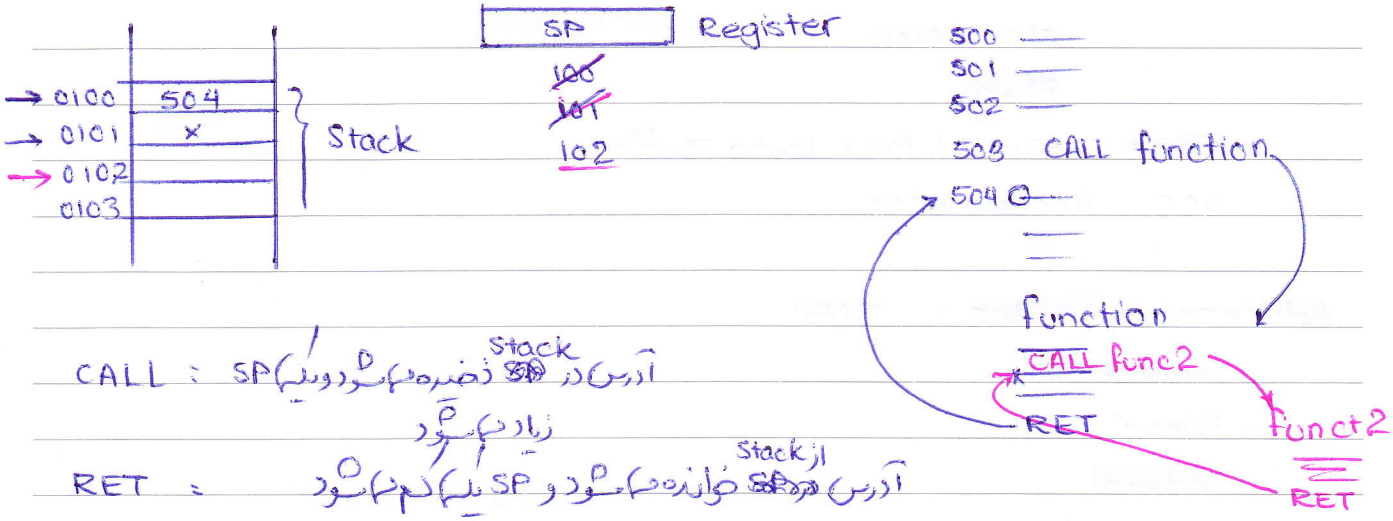
POP A:

SP ← SP - 1

Instructions that change SP : CALL / RET / POP / PUSH / interrupts

SP : نشانگر آدرس Stack را در خود نگه می‌دارد

Stack : آدرس برگشت interrupt را در آن ذخیره می‌کند.



CALL : SP یکی زیاد می‌شود و آدرس بازگشت در Stack ذخیره می‌شود

RET : آدرس بازگشت از Stack خوانده می‌شود و SP یکی کم می‌شود

interrupt : مانند یک CALL عمل می‌کند

Special Function Registers 8051 :

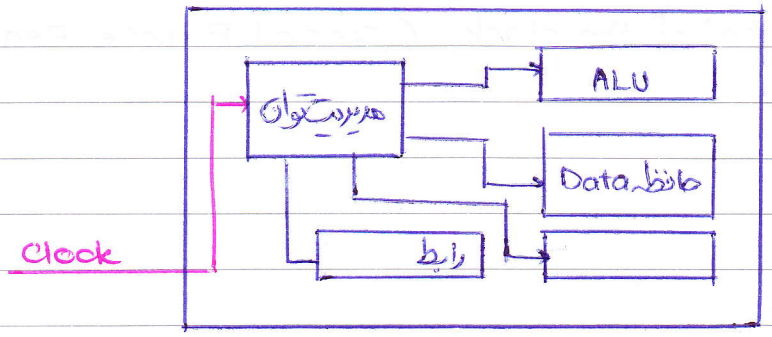
DPL / DPH : DPTR , Data pointer

PCON : Power Control

TCON : Timer Control

TMOD

TL0 / TH0 , TL1 / TH1



مدیریت توان وقتی یکی از واحدها فعال نیست پالس کلاک به آن‌ها فرستاده نمی‌شود و توان کمتری مصرف می‌شود.

SFR (Special Function Register):

SCON: Serial Control
- Baud rate
- start/stop
- Flag

SBUF: Serial Port Data Buffer

ACC: Accumalator

Addressing Modes: 8051
- Immediate
- Direct
- Indirect
- External
- Code

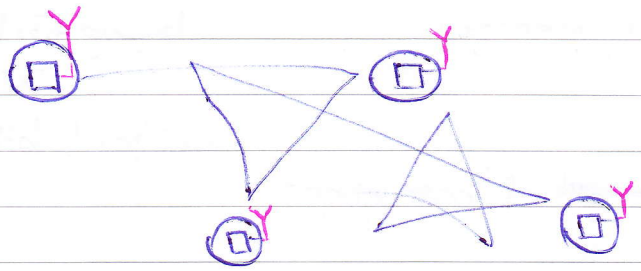
Micro Chip

General Instrument از Spin off

Serial Storage:
- Serial EEPROM
- Serial SRAM

USB Controller:
- ZigBee interfaces

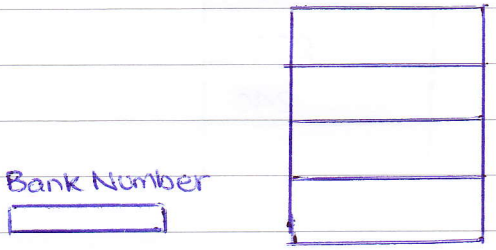
تکنولوژی برای برقراری لینک wireless
لینک‌های ZigBee از یک تعدادی Node تشکیل شده‌اند.



استانداردی برای شبکه‌های wireless با topology نامحفر. این شبکه ZigBee
ساختارش هر روز تغییر می‌کند
(برای اینکه این حسگرها از آن هستند می‌توان برای ارتباط برقرار کردن با کنترل‌ها آنها از این تکنولوژی
استفاده کرد - در هر کنترل یک حسگر قرار دهیم)

Microchip PIC :

- 1975
- PIC : Programmable Interface Controller
  - ~ No difference between : memory space
    register space
    RAM serves as : memory & registers
  - ~ Banking mechanism :



حافظه‌ی 1024 خانه

PIC Micro-controller :

~Later versions :
Move instructions
Capable of addressing whole memory.

با حافظه‌ی خارجی ارتباط ندارد (8051 داشت)

- ~ ROM
- ~ EPROM
- ~ Flash ROM

} عملی هست زیرا باید برنامه را به شرکت سازنده داد تا برای ما بسازد

Erasable Programmable
ROM = EPROM

Subject: 20

89 / 7 / 25

Addressing in program memory

تفاوت اصلی PIC با بقیه چیپ‌ها
طول دستورها ثابت است
حافظه‌ی برنامه از حافظه داده جدا است

~ Depends on length of instructions

Each instruction

~ Takes 2 instruction cycle to execute.

Profiling

بررسی کردن دستی قطعه کدها که چقدر طول می‌کشد اجرا شود.

کوییز: روند تغییرات PC را مشخص کنید.

Program Counter

ORG دستور نیست

SP: ~~100~~ → ~~101~~ → ~~102~~ → ~~101~~ → ~~100~~ → 99

ORG 20

Address	Instruction		Note	Stack addr	Stack value
20	___				
21	___			98	200
22	PUSH	25		99	203
23	CALL	50	stack ذخیره	100	~~220~~ → 25
24			و SP ← SP+1	101	~~230~~ → 24
					235
					240

ORG 50

Address	Instruction		Note
50	___		
51	___		
52	POP	A	
53	POP	A	
54	RET		ابتدا SP ← SP-1 سپس stack را می‌خواند

PC : 20 → 21 → 22 → 23 → 50 → 51 → 52 → 53 → 54 → 203

PIC Performance :

Profiling :

Important in real-time systems

در خودرو مثلاً پردازش باید real-time باشد

پردازش سیگنال عمدتاً [illegible] real-time

Video Camera / Voice-recorder

A program that performs profiling:

profiler : Intel VTune Performance Analyser

(کدهای برنامه شما زمان بیشتری برای اجرا میخواهد را مشخص میکند تا بتوانید برنامه را Optimize کنید)

PIC Compiler:

μOP (میکرو آپریشن) — صفر و یک — μCODE (روی CPU)

OPERATION CODE — صفر و یک

Assembly Program — String / عبارات → Assembler

C Program / Fortran → C Compiler

Visual Programming (Visual Basic)

Visual Programming وقتی به کار برده میشود که هسته ی پردازشی بیش از یکی باشد

2008 :

Microchip announced their own compiler

For 18F , 24F and 30F devices

AVR v.s. PIC :

AVR :

Atmel by GNU C Compiler (خیلی جلوتر از PIC است)

(کامپایلر خیلی کامل)

PIC 16x :

33 fixed length 12bit instructions (RISC)

32 bytes register file

Bank number : تعداد معدودی بیت معدودی خیلی بزرگ را آدرس دهی میکنیم

PIC 17x :

16 bit opcodes

Read access to code memory

External program memory interface (حافظه ی خارجی میتوان)

(آنها اضافه کردند در PIC های جدید ولی در قدیمی این طور نیست)

http://multimedia.iut.ac.ir

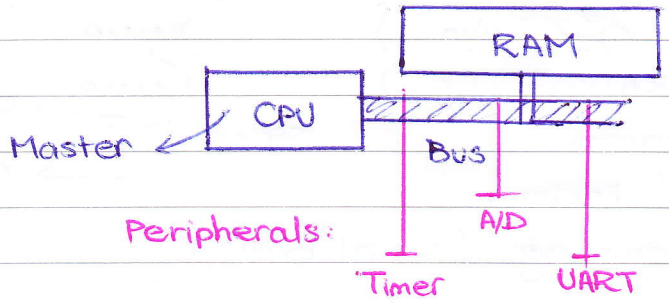
Subject: 22 Firefox → flashgot

dsPIC :

Supported by GNU C Compiler (GCC)

Interrupt Vector Table

DMA : Direct Memory Access



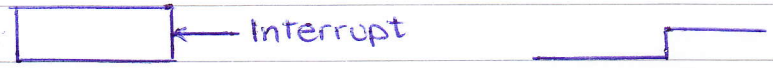
عملیات بر روی باس انجام می‌شود (عملیات انتقال داده Transaction) تحت نظارت CPU انجام می‌گیرد. هر transaction یک سر CPU است. Master CPU باس است.

Read transaction

Write transaction

اما در یک طراحی خوب Access مستقیم است و بدون عبور از CPU، transaction صورت می‌گیرد. اگر DMA باشد سیگنال‌های خود را می‌تواند انتقال داده داشته باشد و Master باشد. در این حالت CPU می‌تواند کارهای دیگر انجام دهد.

Interrupt:



Interrupt Handler / Interrupt Service Routine برنامه‌ای که به هنگام اجرای Interrupt

Interrupt Vector Table:

0	1000
1	
2	
3	

آدرس routine در یک جدول نگه‌داری می‌شود به user اجازه می‌دهد که جدول را به دلخواه خود پر کند

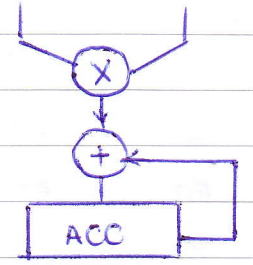
در غیر این صورت به هنگام interrupt به یک مکان مشخصی می‌رود.

پایه‌ی تمام محاسبات (الگوریتم‌های پردازش سیگنال)

$Y = Y + \alpha X$

Multiply

Accumulate



حداقل یکی از این حاد در پردازنده‌های امروزی وجود دارد.

$$y[n] = \sum_{i=0}^{k} h[i]\, x[n-i]$$: Multiply and Accumulate

Intel 8086:

همان CPU که CPU استفاده می‌کنند، Instruction set آن هنوز هم استفاده می‌شود.

16 بیتی

8088 : CPU ای که در اولین PC (Personal Computer) استفاده شد.

عرض باس آن با 8086 متفاوت است.

Backward compatible

8008 , 8080 , 8085

Support for

Full 16 Bit processing

Base + offset addressing

(در PIC با بانک آدرس می‌توانیم از حافظه‌ی بزرگتری استفاده کنیم و آدرس دهی کنیم)

بیس و افست مانند همان بانک آدرس عمل می‌کند (بیس = شماره بانک افست = باقی آدرس)

Self Repeating operations

Micro-coded Multiply and Divide

20,000 transistor

IC , 40 پایه

بسیاری از پایه‌ها دوباره یا چند باره هستند

2^{20} = 1 million byte

حافظه به آن وصل می‌شود.

DRAM , SRAM

معمولاً آدرس می‌گیرد و بعد داده می‌گیرد DRAM

ولی SRAM این طور نیست بنابراین مقداری تاخیر دارد اگر بخواهد با این CPU کار کند

MIN MODE / MAX MODE

اگر فقط CPU 8086 باشد باید در Min mode باشد

Max Mode در صورتی که ...

Subject: 24

Core Duo2 : CPU مورد استفاده از BMW

AMD : Athlon FX Ferrari

Register : 8086 :

Main Registers :

AH	AL	AX	(Primary accumulater)
BH	BL	BX	
CH	CL		
DH	DL		

Index Registers :

SI	Source Index	انتقال داده
DI	Destination Index	
BP	Base Pointer	
SP	Stack Pointer	

Status Register :

بعد از هر دستور ریاضی یا منطقی update می‌شود رجیستر

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	O	D	I	T	S	Z	-	A	-	P	-	C

پایه‌ها :

- NMI None Maskable Interrupt غیرفعال نمی‌شود :
- INTR غیرفعال می‌شود ← توسط بیت I

بیت I فرق دارد وقتی یک می‌شود Interrupt فعال می‌شود.

بیت‌ها :

- O : Overflow
- D : Direction
- I : Interrupt enable
- T : Single Step flag
- S : Sign
- Z : Zero flag
- A : Some kind of carry
- P : Parity
- C : Carry

Subject: 25

| 89 | 8 | 2 |

Segment Register :

برای گسترش حافظه :
حافظه را segment بندی میکنیم و آدرس دهی میکنیم

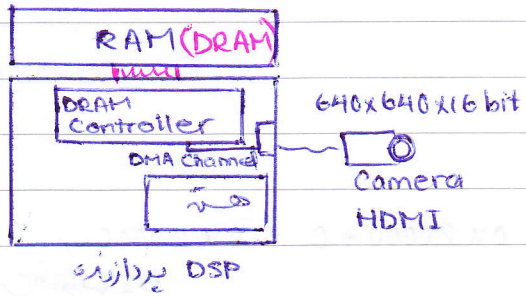
14 ⎧ CS : Code Seg.
⎪ DS : Data Seg.
⎪ ES : Extra Seg.
⎩ SS :

MOV AX, [SI]
DS << 4 + SI آدرس

JMP 0x80
CS << 4 + SI

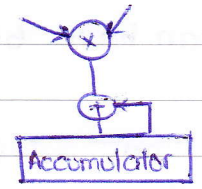
کوئیز : مفهوم واژه های
DMA - : خیلی مهم است (ارتباط peripheral ها - یک مکانیزم است)
MAC -
Interrupt Vector -

DMA :
نمی توان در پردازش تصویر از SRAM استفاده کرد زیرا حجم اطلاعات خیلی زیاد است (RAM = DRAM)



اگر بخواهیم بدون درگیر کردن CPU فریم بعدی را (همزمان با پردازش فریم فعلی) در RAM قرار دهد باید DMA وجود داشته باشد. ذخیره ی فریم بعدی بدون اتمام پردازش فریم اولی شروع میشود.

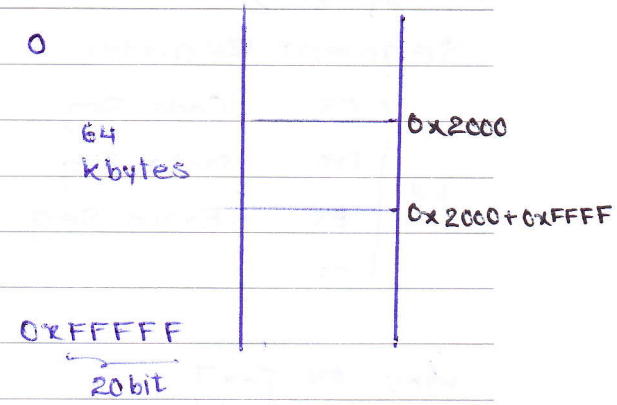
MAC :
پردازش سیگنال - شکل



Interrupt Vector : بخشی از حافظه را اختصاص میدهیم به ذخیره ی آدرس Interrupt routine
وقفه : برای هر وقفه یک برنامه مینویسیم به نام Interrupt Service Routine معمولا این routine ها در interrupt vector ذخیره میشود.

Subject: 26

حافظه‌ی 8086 :



SI : Source Index
یک بلوک داده را از یک جایگاه حافظه‌ای دیگر انتقال دهیم

SI	= 16 bit

DS = Data Segment
DS x 16 + SI آدرس را این طوری تولید می‌کنیم

DS = 0x00
SI = 0

دستور انتقال بلوک :

[SI] ⇒ [DI]
محتوای SI نشان داده / محتوای DI نشان داده

تولید فیزیکی آدرس

[DS : SI] ⇒ [ES : DI]
DS ← Segment ، SI ← Offset

DS = 0x200
SI = 0
→ 16 x 0x200 + 0 = 0x2000 + 0 = 0x2000 آدرس فیزیکی SI

ES = 0x4
DI = 0x60
→ 0x40 + 0x60 آدرس فیزیکی DI

each segment can have 64 kbytes range of addressing

چون می‌خواهیم آدرس‌هایمان از 16 bit بیشتر باشد برای آدرس‌دهی 1Mbyte حافظه باید از این روش استفاده کنیم.

DS : Data
ES : Extra
CS : Code ناحیه‌ای از حافظه به عنوان برنامه دارد استفاده می‌شود.
SS : Stack ناحیه از حافظه به عنوان Stack استفاده می‌شود

DS
ES
CS
SS

Program Counter = Instruction Pointer (IP)

CS : IP

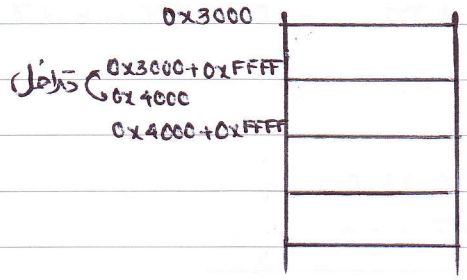
SS : SP (Stack Pointer)

DS = 0x300

ES = 0x 400

CS = 0x 500

SS = 0x 550



0x3000 + 0xFFFF = 0x12FFF > 0x4000

segها تداخل دارند که ایرادی ندارد فقط باید دقت لازم را کرد تا داده‌های مورد نیاز دستکاری نشوند.

در 8086 برای Compile کردن برنامه‌ی C باید در مورد segها راهنمایی کنیم

ویروس‌ها این‌طوری هستند

Tiny ~> segها روی هم (max 64 Kbyte)

Small ~> (128 Kbytes)

Compact

Medium

Large

Huge

Separate Memory and I/O space

64 Kbytes of I/O space

256 interrupts

وقتی وقفه رخ دهد باس منتظر آدرس وقفه‌ها می‌ماند (این مکانیزم با مکانیزم interrupt vector متفاوت است). آدرس را روی Data Bus قرار می‌دهد.

Package ⊗ متفاوت (ظاهر متفاوت)

8087

مکمل 8087 برای اینکه در کنار 8086 کار کند و محاسبات اعشاری را انجام دهد.

Subject: 28

کوییز:
می‌خواهیم یک بلوک داده را از آدرس 0xF3004 به آدرس 0x58008 منتقل کنیم. مقادیر هریک از ثبات‌های DS، ES، SI، DI چقدر باشد؟ حجم بلوک داده چقدر است.

مثلاً:
0xF3004 = DS x 16 + SI = 0xF300 x 16 + 4
0x58008 = ES x 16 + DI = 0x5800 x 16 + 8

ضرب دو عدد:

54 x 32 = 1728

54 ⇒ 110110
32 ⇒ 100000
110110000000

در یک ثبات 8 بیتی می‌خواهد ذخیره کند

1	1	0	1	1	1	1	0

دقت نشان دادن عددها با شدت افت می‌کند.

Mantissa ← $\alpha \times 2^{\beta}$ → Exponent
11011×2^{6}

77 x 35 = 2695

77 ⇒ 1001101
35 ⇒ 100011
10101000000 XXX
000

1	0	1	0	1	1	1	1

Mantissa Exponent

Dynamic Range:
11111×2^{111}
$1 - 31 \times 2^{7}$

$10000 \times 2^{0} = 33$ ⇒ ذخیره | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = 32

ماشتن

یعنی رنج اعداد را بالا بردیم به قیمت از دست دادن دقت

وقتی اعداد به صورت ماشتن و الیشت نمایش داده می‌شوند محاسبات سخت‌تر و عملیات پیچیده‌تر می‌شود تا وقتی که اعداد به صورت معمول نمایش داده می‌شوند.

General Representation of Floating Point

31	30 ... 23	22 ... 0
Sign	Exponent 8 بیت	Mantissa 23 بیت

: IEEE 32 بیتی

17 decimal = 10001 binary

$10001 = 0.10001 \times 2^5$

Then we can construct its representation

0	00101	1000100

تبدیل اعداد اعشاری به اعداد باینری:

$0.23 \times 2 = (0).46$

$0.46 \times 2 = (0).92$

$0.92 \times 2 = (1).84$

$0.84 \times 2 = (1).68$

$0.5 \times 2 = (1).00$ ← اتمام الگوریتم

$0.0011\ldots$

0.0011

$0.1\boxed{1} \times 2^{\boxed{-2}}$

مانتیس — اکسپونت

توان منفی را چگونه ذخیره کنیم؟ برای این طور عدد منفی را (و کلا همه ی اعداد را) قبل از ذخیره باید عدد باینری جمع کنیم تا مثبت شود و سپس ذخیره کنیم. البته این عدد باینری هم باید بتوانیم تعداد بیت دارد.

$0.101 \times 2^{-3} + 0.111 \times 2^{2}$

برای عمل جمع باید ابتدا توان ها را یکسان کنیم.

$0.101 \times 2^{-3} + 111\,00.0 \times 2^{-3} = 111\,00.101 \times 2^{-3}$

$= 0.1110\,0101 \times 2^{2}$

IEEE Floating Point Standard

→ single precision

↘ double precision

Double precision = 64 bits

1 bit = sign

11 bit = exponent

52bit = mantissa

Coprocessors :

A processor that helps main CPU with some tasks

- Floating point arithmetic, signal processing
- Encryption
- Graphics

Provided as a separate component

- To lower prices

سرعت پردازش را با بیش تر کردن تعداد گیت ها توانستند بالا ببرند.

50,000 FLOPS = Floating Point Operations Per Second

8087 : 80 bit registers
8 registers (stack ثبات هایی استک شکل هستند)

کوئیز :

مطلوب است نمایش اعداد 0.5 و 3.14 در فرمت اعشاری دقت مضاعف (32 بیتی)

$3.14 = (11.\quad)_2$

$0.14 \times 2 = 0.28$
$0.28 \times 2 = 0.56$
$0.56 \times 2 = 1.12$
$0.12 \times 2 = 0.24$
$0.24 \times 2 = 0.48$
$0.48 \times 2 = 0.96$
$0.96 \times 2 = 1.92$

$3.14 = 11.001000\,1$
$= 1.100\,10001 \times 2$

ذخیره نمی شود ← ماشتی ↔ اکسپونت

Break Point : نقطه ای که در debug میگذاریم تا CPU آن نقطه بدون هیچ کاضری اجرا کند و سپس منتظر ماند.

Step Over : یک خط اجرا کند

پنجره ی Watch : مقدار variable ها را نشان میدهد.

a = مقدار متغیر   &a = آدرس جایی که متغیرات و ذخیره شده است

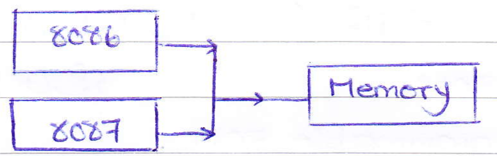
(برعکس خوانده می‌شود)

پنجره‌ی memory : خانه‌ی حافظه را نشان می‌دهد.

در پنجره‌ی memory :   3.14 : 0x 4048F5C3

رجیستر 8086 :   IP   DI   SI   DX   CX   AX

دستورات مخصوص 8087 اولش f دارند مثل   FLD : Float Load



هرگاه دستوراتی که اولش f است به مال 8087 (8087 آنها را اجرا می‌کند و دیگر دستورات را 8086 اجرا می‌کند)

Floating Point instruction : لازم نیست حتماً CPU Float باشد می‌توان خودمان برنامه‌اش را بنویسیم. اگر برنامه‌ای را ببینیم و بخواهیم اصلش را ببینیم از disassembler استفاده می‌کنیم

jmp و jz متفاوتند. jz یک jump شرطی است. ولی Instruction Pointer به معنی اینکه jmp می‌رسد مقدارش را تغییر می‌دهد.

پردازنده‌های DSP :

برای انجام محاسبات با سرعت بسیار بالا

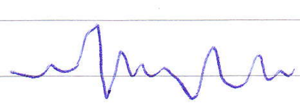
برای خیلی از کارها خوب نیست، مثلاً اگر DataBase داشته باشیم مثل ماشین لباس‌شویی خوب نیست

فرض کنید تلفنی داریم که می‌خواهد پیام صوتی را نگه دارد. برای قابل فهم بودن پیام نمونه‌برداری باید با ریت 128 KHz نمونه‌های 8 بیتی برداریم.

اگر $f_s = 8\,KHz$ ، دقت : 8 bit $\Rightarrow$ $8000 \times 8 = 64000$ bits per second

$100s \times 64\,kb = 6400\,kbit$

هر 1000 kbit حافظه = \$10 $\Rightarrow$ \$60

 → نمونه‌برداری → 64 kbit/s → پردازنده‌ی فشرده‌ساز (DSB) → 8 kbit/s

89 | 9 | 11

کوئیز

0 , 8 , 4 , 12 , 2 , 10 , ?

? = 4 درواقع شمارنده‌ای است که به ترتیب از 0 می‌شمارد اما mirror می‌کند.

ب

درگاه ISA : (قبلاً استفاده می‌شد) انتقال داده از این طریق خیلی ساده بود (موازی) 8 Mbyte

درگاه PCI : (بعد از ISA آمد) پهنای باند خیلی بالاتری داشت (موازی) 130 Mbyte

درگاه AGP : (موازی) Accellerated Graphic Port

PCI Express : (سریال)

ROM : حافظه‌ای غیرفرار (در کارخانه ساخته می‌شود و قابل تغییر نیست)

EROM : با تابش اشعه پاک می‌شود Erasable Programmable ROM

EEPROM : Electrically Erasable Programmable ROM

Flash ROM : بلوک‌بندی شده است. نمی‌توان تک‌تک بیت‌ها دسترسی داشته باشیم و همه‌ی بلوک را پاک می‌کنی و برنامه‌نویسی می‌کنیم و اگر بخواهیم عمل خواندن را انجام دهیم همه‌ی بلوک را می‌خوانیم.

DSP : Digital Signal Processing

SSE : Streaming SIMD Extension

SIMD : Single Instruction Multi Data

MIMD : Multi Instruction Multi Data

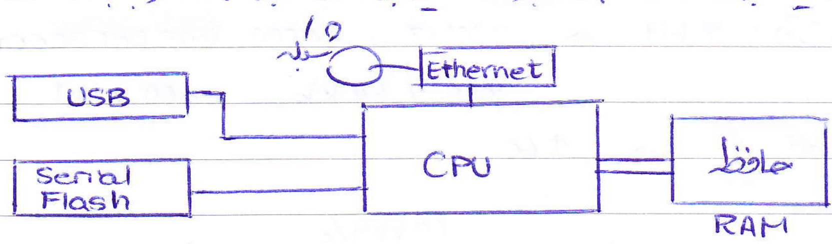
Two Major DSP Categories:

- Floating Point
- Fixed Point DSP : higher level of performance, lower price, smaller

باید حواسمان را جمع کنیم تا داده از دست ندهیم / مدارهای ساده‌تری می‌شود که تعداد بالا رود و همچنین سرعت و کارایی بالاتری (مسائل مربوط به overflow)

Boot Loader:

برنامه‌ی کوچکی که CPU قبل از اینکه برنامه‌ی اصلی را اجرا کند این برنامه را اجرا می‌کند و به بالا آمدن برنامه‌ی اصلی کمک می‌کند.



Boot Loader

Subject: 33

| 89 | 9 | 16 |

Barrel Shifter :

در پردازنده‌ی DSP یک واحد وجود دارد که می‌تواند به هر تعدادی شیفت چپ یا راست دهد. (تا 32 بیت مثلاً)

TMS320C32 : Internal Block Diagram

باس زیاد دارد.

performance محاسبات این خیلی خوب است

یک ALU برای محاسبات + 32 bit barrel shifter

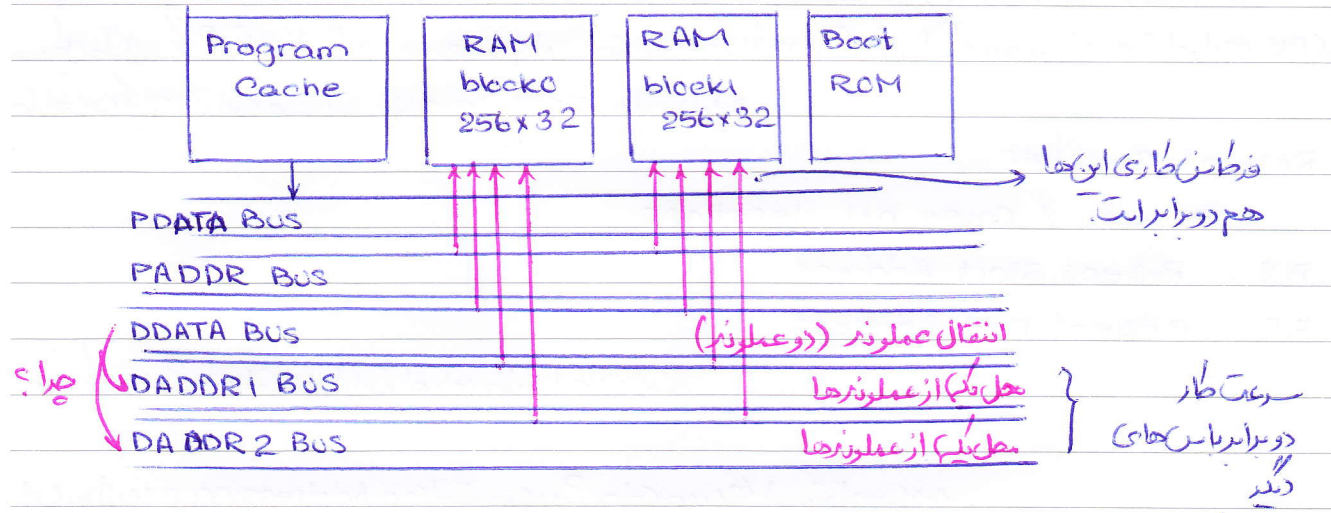
دو ALU برای محاسبه‌ی آدرس (ARAU0 - ARAU1) Auxilary Register Arithmetic Unit

Boot loader : ROM ← در استقان نگاه دارد

DMA Controller (x 2) : انتقال در موازات هسته‌ی پردازنده

↳ متصل به دو باس : DMA Data Bus / DMA Address Bus

↳ متصل به دو باس : Peripheral Data Bus / Peripheral Address Bus



فرکانس کاری این‌ها هم دو برابر است.

مهم است که عملوندها به موقع به واحد پردازنده برسانیم (در performance اساسی است).

Extended precision registers R0-R7 (Floating point)

Auxilary register (محاسبه‌ی آدرس)

Extended Precision Range

Most positive = $(2-2^{-23}) \times 2^{127}$ =

Least positive

Most negative

Least negative

عکس دو ذخیره می‌شود

Register : IF : Interrupt Flag

IF Bits:

INT0
INT1
INT2
INT3
:

} external interrupt flag

Zero Delay Loops:

در برنامه صحتاً از حلقه استفاده می‌شود. یعنی طوری تنظیم شود:

```
for ( i=0 ; i<10 ; i++ )
a++ ;
```

بخشی از زمان CPU صرف چک کردن شرط حلقه می‌شود (نامطلوب ماست). بنابراین حلقه‌ای فوق حداقل 20 سیکل طول می‌کشد تا اجرا شود. اگر بخواهیم این تأخیر برای چک کردن شرط نباشد به صورت سخت‌افزاری CPU را طراحی می‌کنیم. در این صورت Zero Delay حلقه داریم.

Repeat Counter

RC = n / Cause n+1 iterations

RS : Repeat Start Address

RE : Repeat End Address

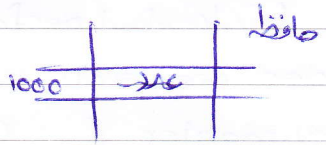
چک کردن شرط حلقه به عهده‌ی اجرای حلقه انجام می‌شود.

این پردازنده در Addressing قوی است پس روش‌های جدیدی استفاده می‌کند.

MPYF * AR2++ , R1

floating point

AR2 | 1000



عدد درخانه‌ی حافظه‌ی 1000 در R1 ضرب می‌شود و در R1 ذخیره می‌شود سپس 1000 به 1001 تبدیل می‌شود. همه در یک سیکل انجام می‌شود.

MPYF * ++AR2(1) , R1

یک جهیز

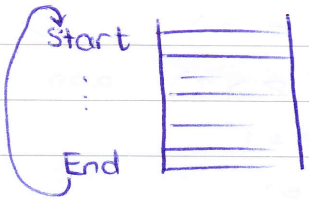
اول محتوای رجیستر AR2 زیاد می‌شود که مقدار این افزایش به عدد داخل پرانتز که خود می‌تواند متغیر باشد سپس ضرب انجام می‌شود.

اگر به + و - باشد مقدار رجیستر تغییر می‌کند فقط به تعدادی که در پرانتز اضافه یا کم می‌کنیم تا آدرس خانه‌ی مورد نظر را پیدا کند ولی محتوای خود رجیستر وقتی تغییر می‌کند که ++ یا -- داشته باشیم.

Subject: 35

89 | 8 | 18

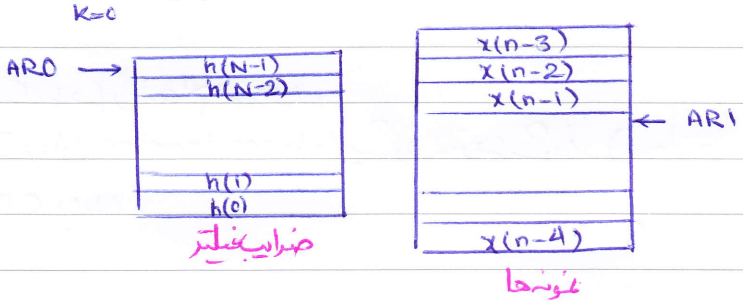
* ARn++ (IR0) B → Bit Reverse Addressing
  % → Circular Addressing



FIR : Filter Implementation

با استفاده از circular addressing

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$



Fetch
Decode
Read
Execute

اگر همه‌ی این کارها را بخواهیم در یک سیکل انجام دهیم زمان یک سیکل خیلی زیاد می‌شود. (100ns)

حالت دیگر این است که در هر سیکل یک مرحله پیش برود. (25ns)
حالت متعارف این است که این واحدها موازی هم دیگر کار کنند نه پشت سر هم.
برای اینکه فرکانس کاری پردازنده بالا برود از pipeline استفاده می‌شود.

Pentium 9 levels in pipeline
:
Pentium4 20 levels in pipeline (فرکانس بالاتر = performance بالاتر)
با بالا بردن تعداد طبقات pipeline فرکانس بالا رفته است تأخیر کمتر

نمی‌توان خیلی تعداد طبقات را بالا برد. یکی علت‌ها این است که تعداد طبقات خیلی نمی‌تواند زیاد باشد این است که اگر دستور branch یا jump اتفاق افتاد در مراحل قبلی که وارد شده‌اند

```
Input → R
R ← R * 1.5
Comp ( R, 3 )
Jump ( R > 3)   BBB
R ← R + 1
Sqrt (R)
Output R
```

تا زمانیکه شرط Jump برقرار است یا نه بنابراین تا Jump برداشته نشود دیگر چیزی وارد pipeline نمی‌شود و کلاً از سطل‌ها هدر می‌رود. هرچه طبقات pipeline بیشتر باشد این وضعیت بدتر می‌شود.

BBB: و همچنین ارتباط CPU با cache اجازه نمی‌دهد تعداد pipeline های level خیلی زیاد شود.

Boot Loader:

وظیفه‌ی مهمی را انجام می‌دهد.
با گذشت زمان boot loader ها پیشرفته‌تر می‌شوند (به طوری که الآن می‌توانند از سیدی یا USB بگیرند و به CPU بدهد).

Branch Operations:

1) Standard branch
2) Delayed branch
3) Branch prediction

- Standard branch: Empty pipeline before performing branch (BR)
- Delayed branch: Doesn't empty pipeline (BRD)

3 دستور بعد از branch را نیز اجرا می‌کند
حسن این branch آن است که pipeline خالی نمی‌شود و هیچ سطلی هدر نمی‌رود.

```
LDF  * + AR1(5), R2
BGED  SKIP          ⇝ conditional delayed branch
LDFN  R2, R1
SUBF  3, 0, R1
NOP                 ; Dummy operation to complete delayed branch
MPYF
 :
SKIP: LDFN
```

(LDFN, SUBF, NOP اجرا می‌شوند)

- Branch prediction: در pentium از این روش استفاده می‌شود.
از دستورات branch آمار می‌گیرد و جواب branch را پیش‌بینی می‌کند.

Subject: 37

1398 | 8 | 23

DSP : چهار یا پنج level بیشتر ندارد

Pentium : delayed branch-Syntax دارد. تعداد pipeline های level زیاد است بنابراین برای آن صلاح است جواب branch را بدانند

14 MFLOPS = 14 میلیون عمل اعشاری

با شرط گذاشتن performance خیلی افت نکند

Code Composer Studio

- Texas Instruments

Simulation : به سخت افزار احتیاج نداریم

Emulation :

In-system-programmable

نرم افزار میکرو را از مدار خارج کنیم و وارد programmer کنیم به رایانه وصل است و برنامه را روی میکرو می ریزد. تنها مزیت این است زیرا می توانیم طراحی میکرو را Monitor کنیم و روی صفحه ی رایانه ببینیم.

Emulation : شبیه سازی / اجرای برنامه داخل مدار (دقیق تر از شبیه سازی)

در حالتی که فضای برنامه اشغال نکند مهم باشد از assembly استفاده می شود (نه زبان C). البته روز به روز compiler ها باهوش تر می شوند و برنامه های compile شده حجم کمتری (از قبل) اشغال می کنند.

C → Compiler → Assembly → Assemble → opcode (object)

obj obj obj

↘ ↓ ↙

linker

↓

exe . com . elf . coff

linker : یک برنامه ی دیگری به برنامه ی ما اضافه می کند و obj. فایل ها را ادغام می کند.

اگر مستقیماً برنامه را به assembly بنویسیم : بهینه است برنامه و حجم کمتری دارد و سرعت بیشتر است. اگر loop داشته باشیم سرعت اجرای آن برایمان مهم باشد (تعداد iteration ها زیادی داشته باشیم)، این طور را می دهیم.

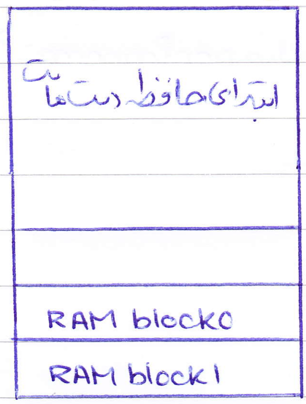
89 | 8 | 25

CPU : Microprocessor Mode : برخلاف boot loader

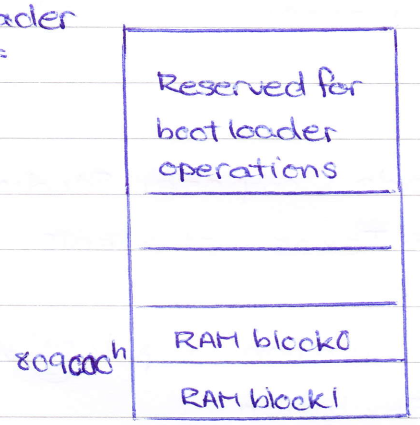
Microcomputer Mode (Boot Loader Mode)

بسته به این دو مد Memory Map فرق دارد.

Microprocessor Mode:



Boot loader Mode :




```
int main (void)
{   int *a = (long *) 0x809C10;
    *a = *a * 2;
}
```

Linker Script:

Specify the Sections Allocation into Memory:

Code .text : > RAM0

initialization Tables .cinit : > RAM0

Constants .const : > RAM0

Systems stack .stack : > RAM1

Variables .bss : > EXT, block 0x10000

.sysmem > RAM1 . یک قسمت از حافظه برای موارد پیش بینی شده تخصیص داده می‌شود.

یک file، text ساده است.

کوئیز :

Linker Script :

RAM0 0x84000000 0x1000

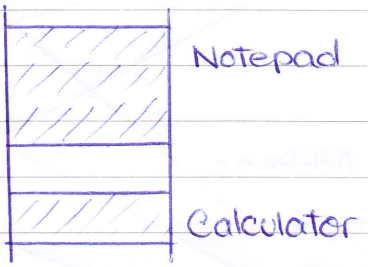
RAM1 0x86000000 0x1000

EXTRAM 0x0

```
.text   EXTRAM
.stack  RAM0
.data   RAM1
.const  RAM1
```

Memory Management Unit :

برای multitasking ، CPU باید از قسمت‌های حافظه‌ی مختلف استفاده کند.



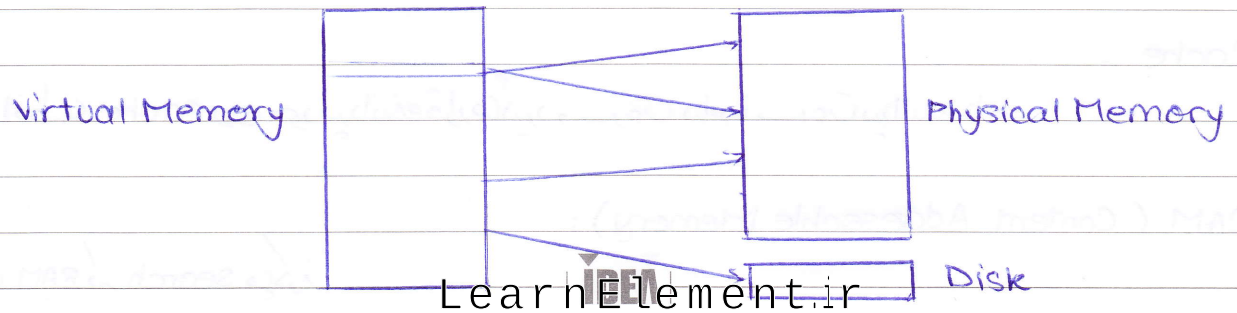
برای debug کردن notepad از آدرس مجازی (مثلاً 0x1001000) استفاده می‌شود.
Applicationها آدرس فیزیکی را نمی‌بینند و از نظر خودشان در آدرس‌های مجازی دارند کار می‌کنند.
MMU این آدرس‌های مجازی را به آدرس‌های فیزیکی تبدیل کند.
این آدرس‌های مجازی در debug کردن (خطایابی) خیلی مفیدند. (آدرس‌ها در همه‌ی کامپیوترها یکسانند)
آدرس‌های مجازی در زمینه‌ی دیگر هم به ما کمک می‌کنند زیرا application ممکن است از فضای Hard disk استفاده کند.

Multitasking Kernel :

- Multiple processes at the same time
- Each process needs its own part of the memory
- Libraries :
  - ~ Static library
  - ~ Shared library

Virtual Memory :

یک mapping بین حافظه‌ی مجازی و حافظه‌ی فیزیکی وجود دارد.



Paged Virtual Addressing:

- We divide the memory and virtual memory into pages.
- We map the pages together (not each address)

Page Table :

Process1 : Virtual Address Physical Address

0x100 ****

0x101 ****

0x102 ****

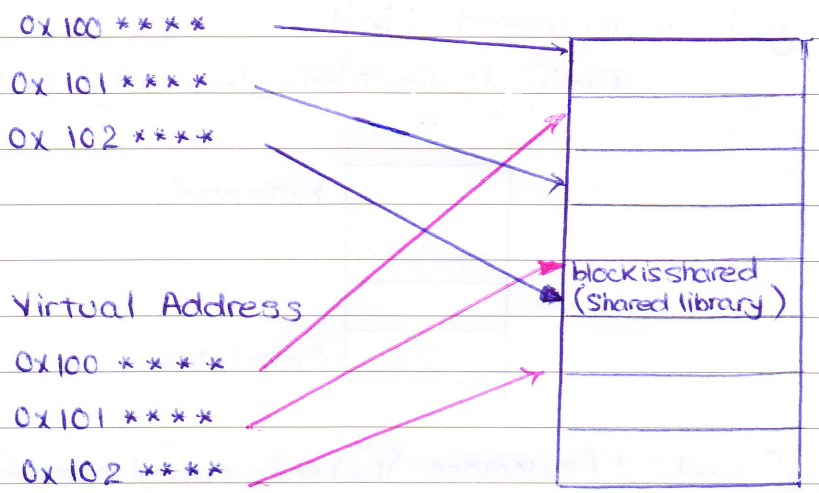
Page Table :

Process 2 : Virtual Address

0x100 ****

0x101 ****

0x102 ****



هر process یک page table دارد. این page table ها را در RAM ذخیره می‌کنیم. (این خیلی بد است زیرا تعداد access ها دوبرابر می‌شود)

Page table can be multi-leveled.

آخرین لحظه که به block access زدیم time stamp می‌خورد. وقتی حافظه کم می‌آوریم از این lastime stamp استفاده می‌شود تا تشخیص داده شود که کدام block مدت بیشتری استفاده نشده. این block به یک منتقل می‌شود تا جا باز شود برای process جدید. اگر مجدداً این block احتیاج پیدا کنیم یک interrupt شروع می‌شود به نام Page fault exception

Translation Lookaside Buffer:

روی خود پردازنده.

page هایی که بیشتر مورد استفاده قرار می‌گیرند روی آن قرار دارند.

سرعت table بسیار بالایی دارد. (TLB یک نوع cache است)

Cache :

حافظه‌ی cache روی خود پردازنده قرار می‌گیرد و سرعت برابر با سرعت پردازنده دارد.

CAM (Content Addressable Memory):

یک RAM که search می‌کند.

Subject: 41

89 | 8 | 30

TLB در واقع یک CAM است (Search می‌کند آیا معادل آدرس فیزیکی آدرس مجازی را دارد یا نه و اگر دارد آن آدرس فیزیکی چیست).

TLB می‌تواند دو تا باشد (Instruction/ Data اگر معماری Harvard داشته باشیم).

A Typical TLB:

- Size: 4096 rates
- Hit time: 1 clock cycle (معادل هزینه)
- Miss penelaty: 100 clock cycle (اگر آدرس مجازی در TLB نباشد)
- Miss rate: %1 (معمولاً آدرس مجازی در TLB هست)

X86 Protected Mode

(Real Mode: آدرس فیزیکی و آدرس مجازی یکسان هستند)

application ها فقط آدرس‌های مجازی را می‌بینند.

- Memory paging
- Larger physical

غیرفرار: hard disk , flash

فرار: RAM

RAM: سریع‌تر از harddisk و flash است و دیتا را به اطلاعات آن آسان‌تر است

harddisk ⟷ RAM ⟷ CPU

باعث افزایش سرعت

داده‌ها و دستورالعمل‌هایی که به آن‌ها بیشتر احتیاج داریم

DRAM: بزرگ و سرعت کم

SRAM: کوچک و گران

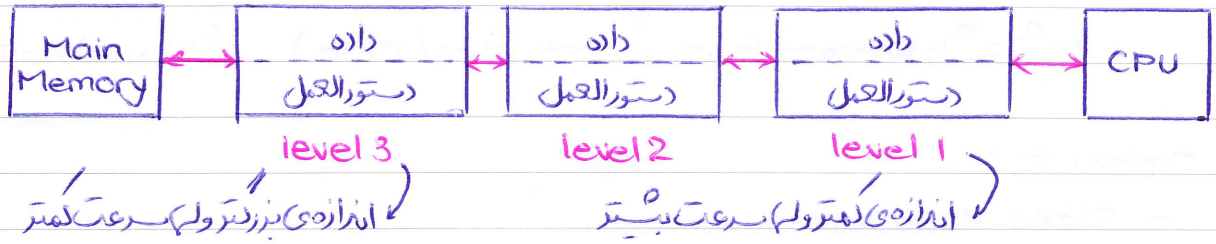
harddisk ↔ DRAM ↔ SRAM ↔ CPU

حجم کم و سرعت بالا (نسبت به DRAM سرعت بیشتر است)

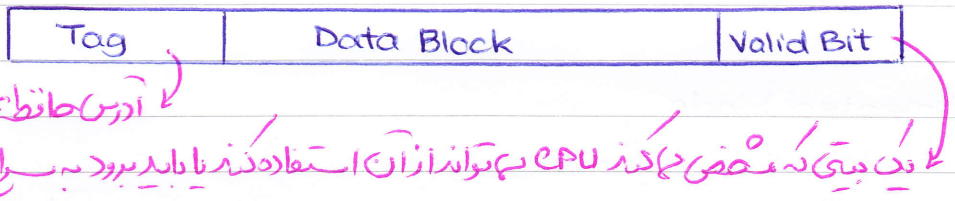
استفاده از SRAM هم برای read و هم برای write استفاده می‌شود. (یعنی انتقال اطلاعات و داده‌ها از دو طرف)

حافظه‌ی Cache :

حافظه‌ای که سریع‌تر و کوچیک‌تر از حافظه‌ی level قبل می‌باشد. زمان دسترسی به حافظه کاهش می‌یابد.
سخت‌افزارهایی که روی سیستم اجرا می‌شوند دارای cache می‌باشند. software , harddisk , CPU



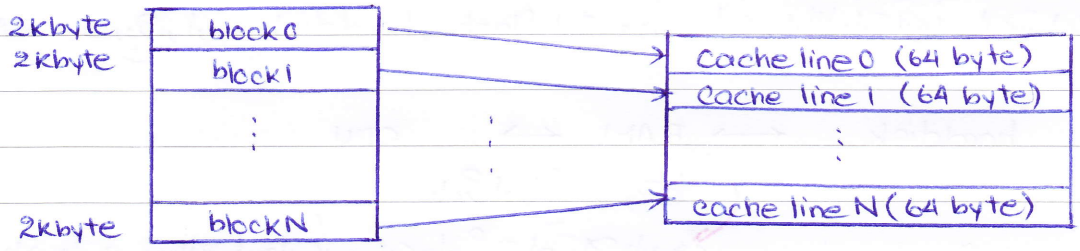
یک خط حافظه‌ی Cache :



ابتدا CPU به Cache دسترسی پیدا می‌کند و در آن search انجام می‌دهد. اگر آدرس موردنظر را داشت که داده به CPU داده می‌شود وگرنه به حافظه‌ی اصلی مراجعه می‌شود و در حافظه‌ی cache ذخیره می‌شود ذخیره می‌شود (تا دفعات بعد سرعت بیشتر داشته باشد)

Direct Mapped Cache

این تکنیک کمترین زمان جست‌وجو را دارد ولی عملکرد آن از همه بدتر است.



heat time خوب دارد ولی head rate آن بد است.

Fully Associative Cache :

مقدار یک خانه‌ی حافظه در هر خانه‌ی دلخواه حافظه می‌توان ذخیره شود و بهتر عملکرد را دارد ولی ساختار این cache پیچیده است. heatrate خوب است ولی heat time آن بد است. (مثلاً TLB)

2 Way Set Associative

یک روش عملی برای map کردن cache و main memory این است که هر خانه‌ی حافظه فقط در دو خانه‌ی cache ذخیره شود و 2 خانه search شود.

Subject: 43

89 | 9 | 2

(ورودی 40) TLB

Data → Cache

(تا 1M بایت) Instruction ↗

دسترسی به DRAM , SRAM : (Access)

SRAM :

مادامی که برق داشته باشد Data روی آن باقی میماند.

DRAM : (Synchronous)

این طور نیست و نیاز به Refresh داریم.

SRAM → Asynchronous

↘ Synchronous

نیاز به پالس ساعت دارد

SRAM : دسترسی 10ns بیشتر طول نمیکشد - قیمت بالایی 100$.

پایه‌هایی دارد که آدرس را مشخص میکند.

16:IO بیت دیتا پاس

Chip Enable : CE

Active Low

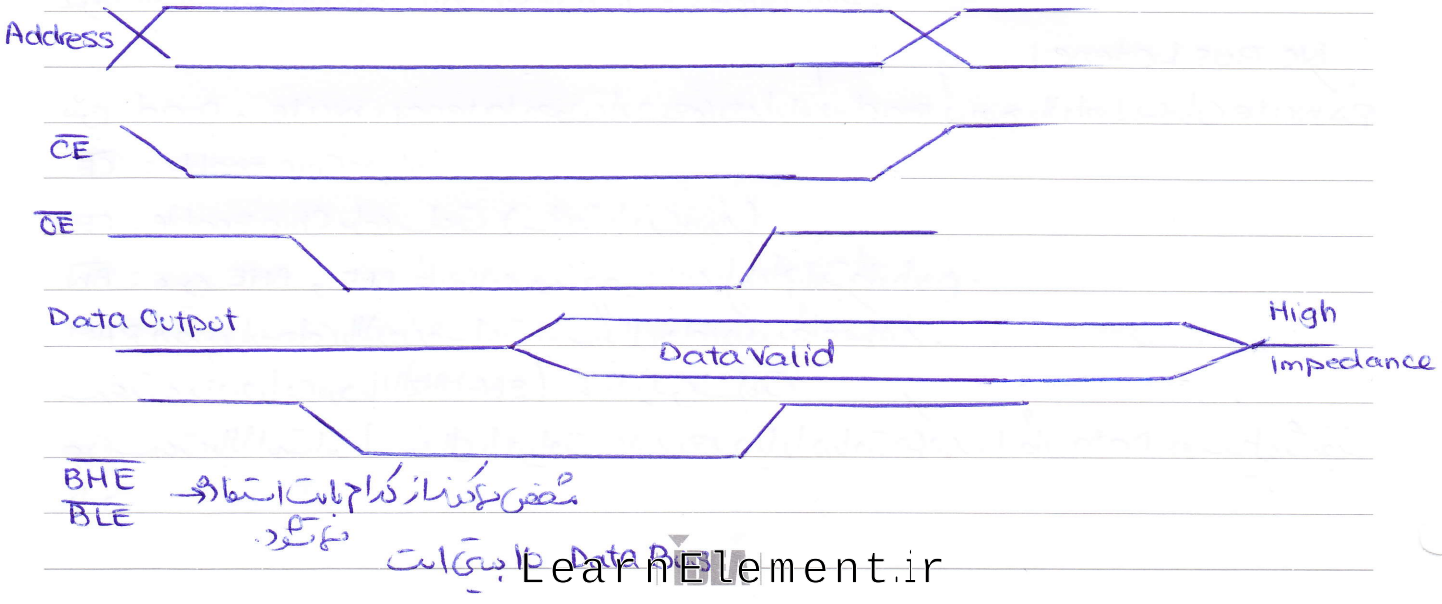
اگر یک باشد چیپ هیچ کاری نمیکند و به حالت high امپدانس میرود (Hi-Z)

انگار که قطع است

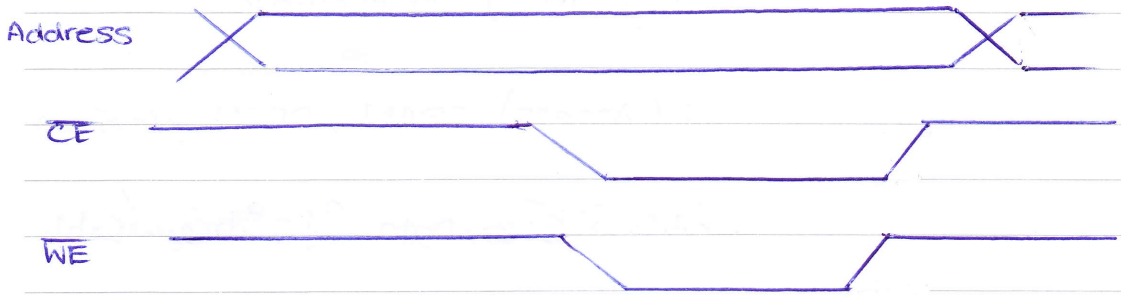
OE : عملکردش مثل CE است یعنی وقتی یک است، خروجی دیده نمیشود. (Output Enable)

WE : باید یک باشد تا خروجی بیرون بیاید (Write Enable)

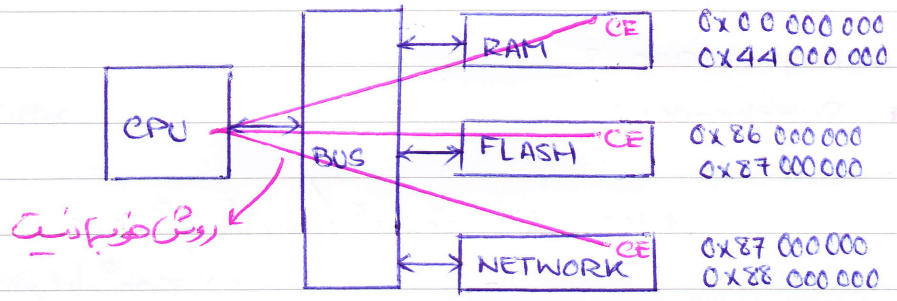
Timing Diagram Read Operation (Asynchronous)



Async- SRAM Write Cycle:



What is chip enable for?



فقط یکی از این‌ها می‌تواند روی باس قرار بگیرد (در هر لحظه فقط یک نفر روی باس قرار گیرد) پس CE هم چه؟
CPU می‌داند که قرار است کارت کند و CE را فعال می‌کند. (این ایده‌ی خوبیه نیست که CPU مستقیماً CE ها را فعال کند زیرا نمی‌داند روی CPU چند پایه‌ی CE قرار باید داد).
کارت بهتر این است که CPU آدرس را روی باس قرار دهد و متناسب آدرس CE مربوطه را فعال شود (Address Decoding)

Synchronous SRAM

عین SRAM آسنکرون است فقط کارهاش با Clock هم‌زمان است.

No Bus Latency:
بین read و write، latency وجود ندارد یعنی بعد از اینکه read کردیم بلافاصله می‌توان write کرد.
CE: Chip Enable طرف است
CEN: Chip Enable برای ساعت (ساعت را نمی‌بیند)
BN: عین BHE و BLE کار می‌کند و مشخص می‌کند با کدام بایت کار داریم.
RAM فقط در لبه‌های بالارونده ساعت به سیگنال‌های دیگر نگاه می‌کند.
سرعت عملکرد را هم ببر بالا (266MHz) کار با پایین ساعت
چون سرعت بالا است ابتدا سیگنال را دریافت می‌کند بعد دیتا را دریافت می‌کند. انتظار Data دو سیکل شیفت دارد.

خاصیت جالب این چیپ آن است که لزوماً آدرس را به آن نمی‌دهیم. آدرس را می‌تواند خودش تولید کند.
اگر CV/LD فعال باشد آدرس را به طور خودکار آدرس را یکی افزایش می‌دهد و اجازه می‌دهد از خانه‌های پشت سر هم استفاده کنیم (burst read/burst write) سؤال امتحانی.

DRAM Memory:

Micron MT46V64M16

Total number of bits : 16bits

4 bank

16M lines

Each line is 16 bits

تمام Operation‌ها با لبه‌ی ساعت کار می‌کند.

: 1 phase

read → 2phase

bank را مشخص می‌کند

یک line در bank را مشخص کنیم

: 2phase

یکی از command‌ها داخل خط را مشخص می‌کند

bank را می‌بندیم

Cast Latency : تأخیرین دستور و قرار گرفتن دیتا روی پایه

دیتاپایه DRAM معمولاً خالی است و طرح طراحی شده read یا write کنیم.
فرکانس کاری بالا ولی باز دهی پایه خیلی کم است (کلاً نمی‌توان سرعت performand را با SRAM مقایسه کرد).

Write Operation:

Write burst

قبل از write باید bank را فعال کنیم (آدرس سطر دارد: شماره‌ی بانک / شماره‌ی خط در بانک / خانه‌ی روی خط)

Acorn :

BBC Micro by Acorn

BBC advertised for BBC Micro widely

Acorn طرفدار گرفت و باید CPU بهتری تهیه می‌کرد و CPU (متفاوت) بسازد مدرنش

در دانشگاه Berkley یک CPU ، RISC ساخته بودند.

Acorn خوشحال شد.

یک CPU بسیار ساده حاصل شد که هیچ تست risky نداشت و در نتیجه power consumption بسیار پایینی داشت

این CPU انقلابی بود و باعث شد که Acorn اسمش را عوض کند.

ARM2 Processor.

Spinoff : Advanced RISC Machines

1992 :

ARM6 : The result of joint efforts by ~~ARC~~ Acorn and Apple

Subject: 1

89/9/16

ARM :

32 bit architecture

Byte = 8 bits
Halfword = 16 bits (2 bytes)
Word = 32 bits (4 bytes)

Most ARMs implement two instruction sets
- 32 bit ARM instruction set — خوب چون خیلی وقت دستورها کوتاه‌ترند
- 16 bit Thumb instruction set

Jazelle cores can also execute Java bytecodes. — هرکدام از دستورها ۸ بیت است در Jazelle. (ARM سخت‌افزار)

CPU می‌تواند switch کند بین مدهای ARM، Thumb، Jazelle. (ابتدا با 32 بیت می‌آید بالا).

Processor Modes :

معمولاً چیزی که applicationها همزمان دارند انجام می‌دهند

ARM has seven basic operating modes :

User : unprivileged mode under which most tasks run.
معمولاً CPU در این مدلت (۸۰٪ موارد).

FIQ :
Fast Interrupt Request — سریع به وقفه جواب می‌دهد

IRQ :
Interrupt Request — تا خیلی جواب به وقفه خیلی مهم نیست

Supervisor

Abort : CPU دست برد می‌زند به قسمتی از حافظه که معلوم نیست (MMU خبر می‌دهد از آن)

Undef : CPU به دستوری که ربطی به Instruction Set آن نیست.

System : privileged mode using the same registers as User mode.

Subject: 2

ARM Registers :

- 16 registers , 32 bits each
- 13 registers are general purpose (بدون تنش هر استفاده دلخواهی از آن میکنیم)

- R13 : Stack pointer
- R14 : LR : Subroutine Linker Register : stores return address
- R15 : Program Counter
  - ARM : دو بیت کم ارزش همیشه صفر است
  - Thumb : یک بیت کم ارزش همیشه صفر است
  - Jazelle : بیت کم ارزش میتواند هرچیزی باشد
- CPSR : Current Program Status Register
  Contains condition code flags
- SPSR : Saved program status Register
  A copy of CPSR

CPSR :

Condition code flags

- N : نتیجه محاسبه ریاضی منفی
- Z : نتیجه محاسبه ریاضی صفر
- C : رخ دهد Carry
- V : رخ دهد overflow

J : Jazelle State

T : Thumb State

Mode bits : specify the process mode

ARM Register Set :

مدهای مختلف برای خودشان shadow register دارند برای استفاده از رجیسترهای اصلی تغییر نکرد در آن مد. برای رجیسترهایی که Shadow ندارند باید ذخیره در حافظه صورت بگیرد (back up).
هرمدی برای خودش SPSR دارد.

Note : system mode uses the user mode registers

Thumb State : فقط ۸ تا از ۱۶ رجیستر استفاده کرد

Exception Handling:

When exceptions occur the ARM:

- Copies CPSR into SPSR_<mode>
- Changes CPSR
- Stories the return address in LR_<mode>

Byte Ordering:

ARM operates in two modes : Big Endian (بایت با ارزش‌تر در محل کم ارزش‌تر)
Little Endian

Instruction Sets :

MOV<cc><S> Rd, <operands>

b2 ↓

flag را تغییر بدهیم یا نه

MOV C S R0, R1 @ if carry is set then R0 := R1

MOV S R0, #0 @ C, V unaffected مقدار صفر را در R0 می‌ریزد.
@ Z=1 , N=0

ADD R0, R1, R2 @ R0 = R1 + R2
ADC R0, R1, R2 @ R0 = R1 + R2 + C
SUB R0, R1, R2 @ R0 = R1 - R2
SBC R0, R1, R2 @ R0 = R1 - R2 + C - 1
(RSB R0, R1, R2 @ R0 = R2 - 1) معکوس

Bitwise Logic:

AND R0, R1, R2 @ R0 = R1 and R2
OR @ or
EOR @ xor
BIC @ R0 = R1 and (~R2) بیت‌هایی از R2 که یک است را در R1 صفر می‌کند

Subject: 4

|89|9|21|

```
CMP  R1, R2        @ set cc on R1-R2   : compare
  (compare)          فلگ‌ها را تغییر می‌دهد ولی مقادیر رجیسترها ثابت می‌مانند.
CMN                @  "   "   "  - R1+R2
  (compare negated)
TST                @  "   "   "  R1 and R2
  (bit test)
TEQ                @  "   "   "  R1 xor R2
```

* Logical Shift left:

```
MOV  R0, R2, LSL #2   @ R0 := R2<<2
                      @ R2 onchanged
```

* Logical Shift Right

```
MOV  R0, R2, LSR      @ R0 := R2>>2
```

* Arithmetic Shift Right

اگر عدد مثبت باشد این شیفت با شیفت منطقی یکی است ولی برای اعداد منفی این طور نیست.

```
MOV  R0, R2, ASR #2   @ R0 := R>>2
```

* Rotate Right

```
MOV  R0, R2, ROR #2
```

شیفت می‌تواند همزمان با اعمال ریاضی انجام شود.

64 Bit Addition

```
ADDS  R2, R2, R0
ADC   R3, R3, R1
```

کوشش:

مقدار ثبات R0 را در 37 ضرب می‌کنیم:

$37\,R0 = 32R0 + 4R0 + R0$

```
MOV   R1, R0, LSL #2
MOV   R2, R0, LSL #5
ADD   R3, R1, R2
ADD   R0, R0, R3
```

Multiplication :

```
MUL   R0, R1, R2        @ R0 = R1 x R2
```

- R2 can't be immediate
- R0 and R1 can't be the same

```
MLA   R4, R3, R2, R1    @ R4 = R3 x R2 + R1
                        @ Multiply and accumulate
```

Multiply with a constant can often be more efficiently implemented using shifts

```
MOV   R1, #35
MUL   R2, R0, R1
```

or

```
ADD   R0, R0, R0, LSL #2    @ R0' = 5 x R0
RSB   R2, R0, R0, LSL #3    @ R2 = 7 x R0' = 8R0' - R0'
                            @ Reverse Subtract
```

MUL , MLA زمان بیشتری طول می‌کشد تا اجرا شود. (به دلیل ساختار سخت‌افزاری)

Data Transfer Instructions :

Moving data between registers and memory.

Three basic forms

- Single register load/store
- Multiple register load / store
- SWP (swap)

```
LDR   R0, [R1]      @ R0 := mem32[R1]
STR   R0, [R1]      @ mem32[R1] := R0
table:  .word 10
ADR   R0, table
```

Addressing Modes:

Three ways to specify offset:

~ Constant

```
LDR R0, [R1, #4]      @ R0 := mem[R1+4]
```

خود R1 تغییر نمی‌کند اگر بخواهیم تغییر کند باید ! داشته باشد

~ Register

```
LDR R0, [R1, R2]      @ R0 := mem[R1+R2]
```

~ Scaled

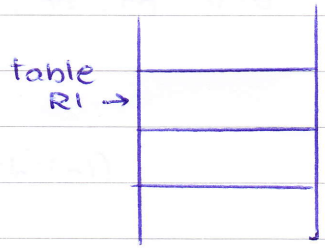
```
LDR R0, [R1, R2, LSL #2]
```

```
LDR R0, [R1, #4]!     @ R1 = R1+4 , R0 := mem[R1+4]
                      @ No extra time, fast
```

Pre-indexed addressing / Post-indexed addressing

(بعد از اینکه access انجام شد R1 را تغییر می‌دهیم)

```
loop:  ADR  R1, table
```



LDM → load multiple registers / STM → Store multiple registers

```
LDMIA  R1, {R0, R2, R5}
```

Subject:

suffix :

IA	increase after
IB	increase before
DA	decrease after
DB	decrease before

LDMIA R_n, {R_1, R_2, R_3}

LDMDA R_n, {R_1, R_2, R_3}

LDMIA R0, {R1 - R3}

R1 : 10
R2 : 20
R3 : 30
R0 : 0x10

add	data
0x00	10
0x04	20
0x08	30
0x0C	40
0x020	50
0x024	60

LDMIA R0!, {R1 - R3}
R1 : 10
R2 : 20
R3 : 30
R0 : 0x0C

LDMIB R0!, {R1 - R3}
R1 : 20
R2 : 30
R3 : 40
R0 : 0x0C

LDMDA R0!, {R1 - R3}
R1 : 40
R2 : 50
R3 : 60
R0 : 0x018

LDMDB R0!, {R_1 - R_3}
R1 : 30
R2 : 40
R3 : 50
R0 : 0x018

Subject:

Application : Copy a block of memory (32 bytes/aligned)

```
loop :   LDMIA   R9! , {R0-R7}
         STMIA   R10! , {R0-R7}
         CMP     R9, R11
         BNE     loop          @ branch not equal
```

Branch Conditions :

	B BAL	
*	BEQ	Equal
*	BNE	Not equal
	BPL	Plus (result was positive or zero)
	BMI	Minus
	BCC	Carry clear
*	BGT	Greater than
	BGE	Greater or equal
*	BLE	Less or equal

Branch and Link :

BL instruction saves the return address to R14 (LR) → Linker Register

```
   BL   sub
    :
sub: - - -
     :
   MOV  PC, LR
```

Stack :

Mode	LDM (POP)		STM (PUSH)	
Full Ascending (FA)	LDMDA	LDMFA	STMIB	STMFA
Full Descending (FD)	LDMIA	LDMFD	STMDB	STMFD
Empty Ascending (EA)	LDMDB	LDMEA	STMIA	STMEA
Empty Descending (ED)	LDMIB	LDMED	STMDA	STMED

Subject:

89 / 9 / 23 9

ARM9 : DRAM Controller ⇐ DRAM را به میکرو وصل می‌کنیم

64 Mbyte حافظه

داراست : حافظه‌ی flash

Slot دارد برای اضافه کردن flash

USB → host

→ peripheral

تمرین:

یک برنامه‌ی Assembly بنویسید که 64 داده 4 بایتی را از آدرسی که R0 مشخص کرده به آدرسی که R1 مشخص کرده، انتقال دهد.

* ARM Real View Development Suit. (RVDS)

Assembler / Compiler. مبتنی بر eclipse است.

* Keil MDK

* IAR Embedded Workbench

* Green Hills

* GCC : GNU C Compiler : Open Source رایگان.

از رنج خیلی بزرگ CPU حمایت می‌کند از جمله x86 , Power PC , ARM , AVR , PIC

انجام می‌دهد cross compilation

برنامه‌ی ARM را کامپایل می‌کنند چون روی platform x86 است پس cross compile کرده‌ایم.

* Windows CE از شرکت Microsoft

Watchdog Timer : به صورت سخت‌افزاری می‌شمارد و وقتی به یک Threshold رسید reset می‌کند (برای مواردی که سیستم hang می‌کند) می‌توانیم آن را disable می‌کنیم. ما هر چند وقت یک بار آن را صفر می‌کنیم تا reset نشود ولی اگر سیستم hang کرده باشد میکرو دیگر صفر نمی‌شود بنابراین پس از مدتی reset می‌شود میکرو.

وقتی میکرو بالا نوشته شد باید initialization انجام شود و سپس کد مربوط را در برنامه قرار دهیم.

برنامه نویسی به اسمبلی

```
__asm MyFunc ( unsigned int * srcAddr, unsigned int * dstAddr)
{ STMDB   R13!, {R0-R10}        ----> MOV R9, #16
label
  LDMIA   R0!, {R4-R7}
  STMIA   R1!, {R4-R7}          label = MyFunc_loop
  SUBS    R9, #1  ~~> R9, R9, #1
  BNE     MyFunc_loop
  LDMIA   R13!, {R0-R10}
  MOV     PC, LR  ~~> BX <rn>
}
```

flag ها set شوند

```
int main (void)
{ int i;
  unsigned int srcArray[64], dstArray[64];
  for(i=0, i<64, i++) srcArray[i] = i*3;
  MyFunc(srcArray, dstArray)
}
```

Branch & Link

```
BL  sub1        @ call sub1
```

زیرا در sub دوباره LR وجود دارد

```
sub1 : STMFD R13!, {R0-R2, R14}
       BL sub2
       ...
       LDMFD R13!, {R0-R2, PC}
sub2 : ...
       ...
       MOV PC, LR
```

Conditional Execution:

```
      CMP R0, #5
      BEQ bypass      @ if(R0 != 5)
      ADD R1, R1, R0  @ R1 = R1+R0-R2
      SUB R1, R1, R2
bypass:
```

Subject:

این برنامه معادل برنامه‌ی قبلی است و branch دارد ولی این برنامه کوتاه‌تر و سریع‌تر است

```
CMP     R0, #5
ADDNE   R1, R1, R0
SUBNE   R1, R1, R2
```

Rule: if the conditional sequence is three instructions or less we use conditional execution.

مثال دیگر:

```
if ((R0 == R1) && (R2 == R3)) R4 ++
```

Instruction Set:

BIC : Bit Clear

RSB : Reverse Subtract (دوم منهای اول)

SUB : Subtract (اول منهای دوم)

TEQ : Test Equivalence (از حاصل XOR فلگ‌ها را تست می‌کند)

TST : Test (از حاصل AND فلگ‌ها را set می‌کند)

CMP : Compare (می‌توانیم عددها را با CMP انجام دهیم)

ARM assembly program

label	operation	operand	comments
main:	LDR	...	@
↓	STR	...	
الزامی نیست	SWI	#11	

```
ADR R0, value1   ~>   ADD R0, PC, #40
```

```
value1:  .word  0x00000001, 0xF0000000
value2:  .word  0x00000000, 0x10000000
result:  .word  0
```

Loops :

for(i=0 ; i<10 ; i++) {a[i]=0;}

```
        MOV   R1, #0
        ADR   R2, a      ← آدرس a را در R2 بفرست
        MOV   R0, #0     ← i همان
LOOP:   CMP   R0, #10
        BGE   EXIT
        STR   R1, [R2, R0, LSL #2]
        ADD   R0, R0, #1
        B     LOOP
EXIT:   ...
```

while loops :

```
LOOP : ...
       BEQ   EXIT
       B
EXIT
```

Find Larger of Two Numbers :

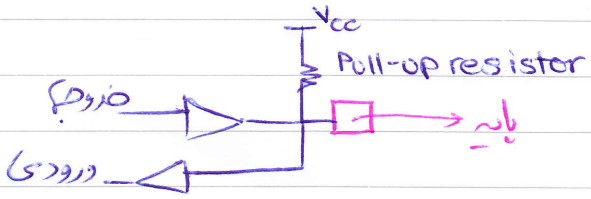
```
        LDR   R1, value1
        LDR   R2, value2
        CMP   R1, R2
        BHI   Done
        MOV   R1, R2
Done:
        ...
```

i-=j بیانگر زمانی است که

ما برنامه را همیشه به C می‌نویسیم و فقط ظاهراً وقات لازم می‌شود که از assembly استفاده کنیم.
(مثلاً وقتی که حساسیت زمانی داریم و می‌خواهیم Optimum نوشته شود یا وقتی که به یک رجیستر خاصی می‌خواهیم دسترسی پیدا کنیم.

Parallel Input/Output Controller :
peripheral که اجازه می‌دهد هر مقدار داده‌ای را روی پایه‌ها قرار دهیم.

buffer حالت
توسط رجیسترها طرز کار آن مشخص می‌شود



Pull-up : وقتی پایه درایو نشده پایه حتماً یک (یا Pull-down، صفر) تولید می‌کند

Glitch Input Filter Enable Register : تغییرات سریع سیگنال Glitch
که معمولاً نامطلوب هستند فیلتر شوند.