Computer Organization And Microprocessors ENCS2380

By :

Mohammed Saada

Performance

$$\boxed{\text{Performance} = \frac{1}{\text{Excution time}}}$$

$$\boxed{\begin{aligned}\text{Cpu Excution time} &= \text{cpu cycles} \times \text{cycle time} \\ &= \frac{\text{cpu cycles}}{\text{clock rate}}\end{aligned}}$$

clock cycle = clock period

$$\text{clock rate} = \text{clock frequency} = \frac{1}{\text{clock cycle}}$$

$$\boxed{\text{CPI} = \frac{\text{total Number of Cycles}}{\text{total Number of instructions}}}$$

CPI : Average cycles per instruction

$$\boxed{\text{cpu cycles} = \text{CPI} \times \text{instructions Count}}$$

$$\boxed{\text{cpu Excution time} = \text{CPI} \times \text{inst. Count} \times \text{cycle time}}$$

MIPS : Millions instructions per second.

$$\boxed{MIPS = \frac{\text{Instructions Count}}{\text{Excution time} \times 10^6} = \frac{\text{clock rate}}{CPI \times 10^6}}$$

$$\boxed{\text{Excution time} = \frac{\text{Inst. Count}}{MIPS \times 10^6} = \frac{\text{Inst. Count} \times CPI}{\text{clock rate}}}$$

① Doesn't take into account the capability of instructions (e.g. Instructions count)

② MIPS varies between programs on the same Computer.

③ MIPS Can Vary inversely with performance

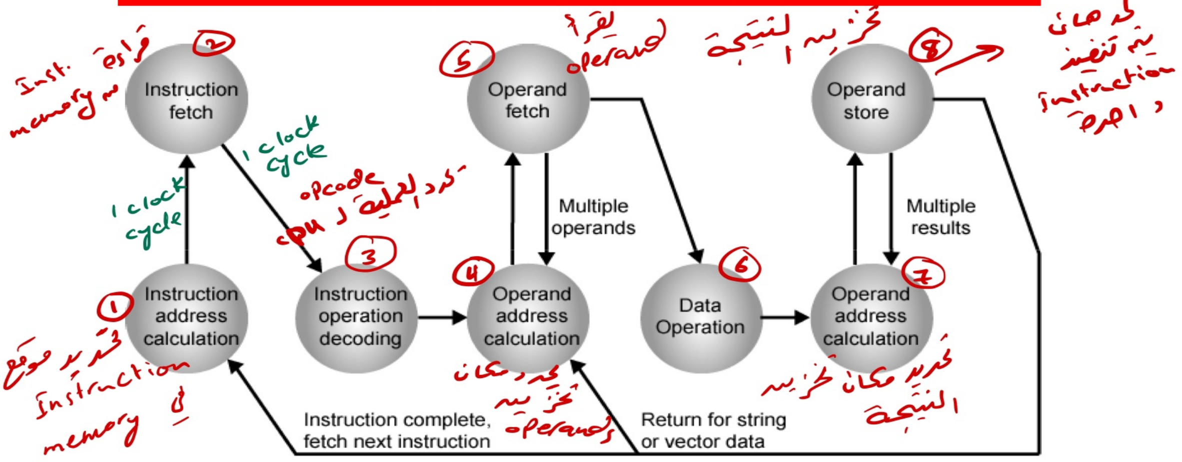
(e.g. a higher MIPS rating does not always mean better performance)

Chapter 10 : Instruction Set Architecture (ISA)

Instruction Format , like :

Opcode	Destination	Source 1	Source 2

Instruction Cycle State Diagram



Instruction fetch
Instruction address calculation
Instruction operation decoding
Operand address calculation
Operand fetch
Multiple operands
Data Operation
Operand address calculation
Operand store
Multiple results
Instruction complete, fetch next instruction
Return for string or vector data
1
2
3
4
5
6
7
8
قراءة Inst. من memory
تحديد موقع Instruction في memory
1 clock cycle
1 clock cycle
تحديد العملية لـ cpu opcode
لقراءة operand
تحديد مكان تخزين operands
تخزين النتيجة
تحديد مكان تخزين النتيجة
لهون يتم تنفيذ Instruction واحد

⇒ Number of addresses (operands) :

① 4 - addresses :
operand 1, operand 2, Result,
next instruction
- very long instruction
- not common.

② 3 - addresses :
operand 1, operand 2, Result.
- ADD A, B, C ; A = B + C

③ 2 - addresses :
operand 1 and Result are same,
operand 2.
- ADD A, B ; A = A + B
- Reduce the length of instruction.
- Some extra work
(to hold Some results if needed)

④ 1 - address
implicit second address (accumulator)
- ADD B ; AC = AC + B
- Common on early machine.

⑤ 0 - address
- using Stack.
- the operands are the top two elements in the Stack, and the Result stored in the stack.

⇒ Byte Order :

① **Little-Endian :**
The least Significant Byte in the lowest address.

② **Big _ Endian :**
The most Significant Byte in the lowest address.

Example : Store 0x12345678 in Byte-addressable memory .

0	0x78
1	0x56
2	0x34
3	0x12
4	

Little Endian

0	0x12
1	0x34
2	0x56
3	0x78
4	

Big Endian

Chapter 11: Addressing Mode

① Immediate (Constant)

- operand = Address field.
- No need to access memory or Registers
- Fast
- Limited
- Leads to poor programming practice

② Direct Addressing (Direct memory)

- Effective address (EA) = address field
- Limited address space ADD A, [5]

③ Indirect Addressing (Indirect memory)

- Effective address (EA) = the content of the memory cell which addressed in the instruction. ADD A, [[5]]
- Large address space.
- Very slow.

④ Register Addressing (Direct)

- Effective address (EA) = R
- Limited number of Registers
- very small address field needed:
  * Shorter instructions
  * Faster fetch.
- No memory access.
- very fast excution.

⑤ Register Indirect Addressing

- Effective address (EA) = the content of the Register.
- operand in memory, it's address in the Register.
- Large address space (2^n)
  n: Register width.
- one fewer memory access than indirect addressing (memory indirect)

⑥ Displacement (indexed) Addressing

- EA = A + (R)
  A: base Value
  R: Register that holds displacement
  or vice versa.
- EA = A + (PC) PC Relative

⑦ Stack Addressing

- operand is implicitly on top of Stack.

Chapter 3: CPU Organization

⇒ The major components of CPU :

1. Register set (Register File)
2. Arithmetic and Logic unit (ALU)
3. Control unit (CU)

⇒ Special purpose Registers in CPU:

PC : program counter.
IR : Instruction Register.
MAR : Memory address Register.
MBR / MDR : Memory buffer (data) Register.
I/O AR : Input/output address Register
I/O BR : Input/output buffer Register.

* RTL Statements (Register Transfer Language)
Micro-operation :

→ Fetch instruction :

$MAR \leftarrow PC$; $PC = PC + 1$
$MBR \leftarrow Mem[MAR]$
$IR \leftarrow MBR$

→ excute the instruction (ADD R_2, R_1, R_0)
the same steps for fetch.

excute :

$R_2 \leftarrow R_1 + R_0$

Example : write the micro-operation for these instructions:

- ADD [X] , R_0 ; Mem[X] = Mem[X] + R_0

Fetch : MAR ← PC ; PC = PC + 1
MBR ← Mem [MAR]
IR ← MBR

excute : MAR ← X
MBR ← Mem [MAR]
MBR ← MBR + R_0
Mem [MAR] ← MBR

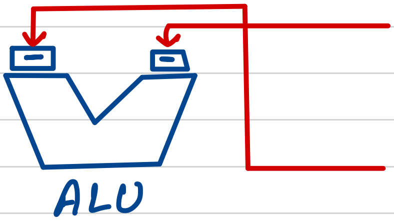
- ADD [R_0] , [R_1]

Fetch : MAR ← PC ; PC = PC + 1
MBR ← Mem [MAR]
IR ← MBR

excute : MAR ← R_1
MBR ← Mem [MAR]

MAR ← R_0
MBR ← Mem [MAR]

MBR ← A + B
Mem [MAR] ← MBR

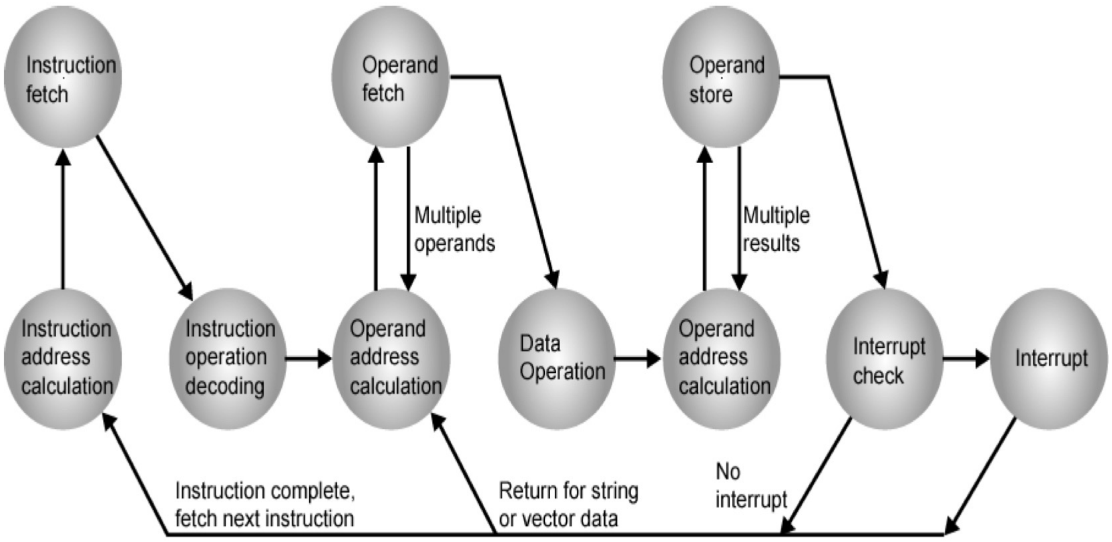


⇒ Interrupts : Mechanism by which other modules (e.g. I/O) my interrupt normal sequence of processing. Improves process efficiency

⇒ Classes of interrupts :

- program : Arithmetic overflow, division by zero
- Timer : Generated by internal processor timer used in pre-emptive multi-tasking
- I/O (from I/O controller) : to signal normal completion or error
- Hardware failure : memory parity error, power failure.

Instruction Cycle (with Interrupts) - State Diagram


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

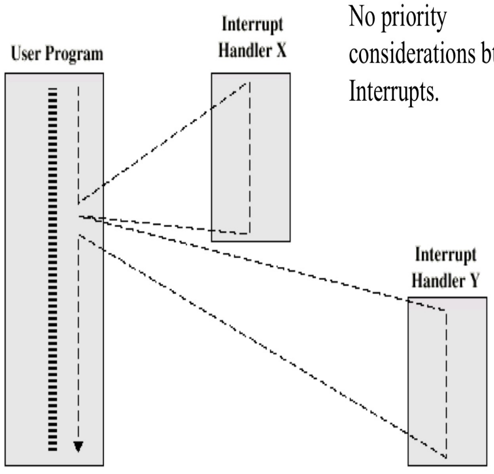
Multiple Interrupts

- Disable interrupts
  - —Processor will ignore further interrupts whilst processing one interrupt
  - —Interrupts remain pending and are checked after first interrupt has been processed
  - —Interrupts handled in sequence as they occur

Queue (First in – First out)

- Define priorities
  - —Low priority interrupts can be interrupted by higher priority interrupts
  - —When higher priority interrupt has been processed, processor returns to previous interrupt

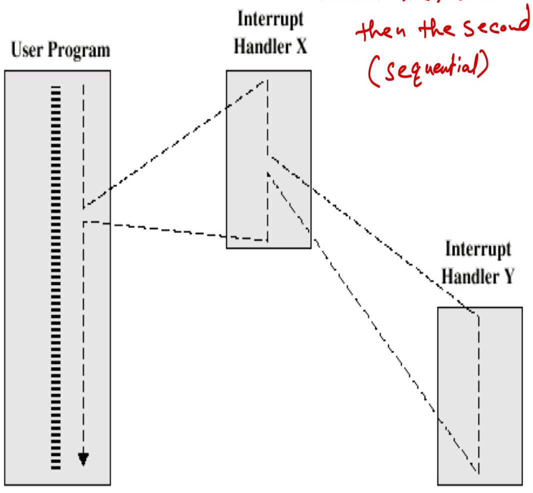
Multiple Interrupts - Sequential



No priority considerations btw Interrupts.

Multiple Interrupts – Nested

if the same priority Process the first one then the second (sequential)



⇛ Bus Types :

① Dedicated : Separate data & address lines.

② Multiplexed : - Shared lines.
- Address Valid or data Valid Control line.
- advantage : Fewer lines.
- disadvantage :
  - More complex control
  - Ultimate performance

⇛ CPU Local (internal) Bus Organization :

One-Bus Organization

- CPU registers and ALU, use single Bus to move outgoing and incoming data.
- Single data movement withen one clock cycle.
- Additional registers may be needed to buffer data for the ALU.
- Simplest and least expensive.
- Limits the amount of data transfer that Can be done in the Same clock cycle.
- Slow down the overall performance.

Two-Bus Organization

- two different Data (operands) can transfered at the Same clock cycle.
- an additional buffer register may be needed to hold the output of the ALU when the two buses are busy (carrying two operands)

- or, one of the buses may be dedicated for moving data into registers (in-bus), while the other is dedicated for transfering data out of the register (out-bus)

Three-Bus Organization

- Two buses used as Source buses, while the third is used as destination.
  - Source buses move data out of the registers (out-bus)
  - destination bus move data into a register (in-bus)
  - each of the two out-buses is connected to an ALU input.
  - The output of the ALU is connected directly to the in-bus.

* More buses → More data can be moved withen a single clock cycle.

* More buses → More complexity in hardware.

⇒ Bus Arbitration

- More than one module Controlling the bus (e.g. CPU, Direct Memory Access DMA)
- Arbitration Can be :

Centralized : only one module may control bus at one time.

Distributed : More than one module controlling the bus (e.g. CPU and DMA controller)

Control Unit

- The control unit is the main component that directs the system operations by **sending control signals** to the datapath.
- ***Datapath:*** The data section, which contains the registers and the ALU.
- These signals control the flow of data within the CPU and between the CPU and external units such as memory and I/O.
- Control buses generally carry signals between the control unit and other computer components in a clock-driven manner.
- The system clock produces a continuous sequence of pulses (timing signals) in a specified duration and frequency.

Control Unit

- A sequence of steps t0 , t1 , t2 , . . . , (t0 < t1 < t2 , . . .) are used to execute a certain instruction.
- The op-code field of a fetched instruction is decoded to provide the control signal generator with information about the instruction to be executed.
- Step information generated by a **logic circuit module** is used with other inputs to generate control signals.
- The signal generator can be specified simply by a set of **Boolean equations** for its output in terms of its inputs.

- There are mainly two different types of control units:
  - Microprogrammed
    - The control signals associated with operations are stored in special memory units inaccessible by the programmer as control words.
  - Hardwired
    - Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

Hardwired Implementation

- In hardwired control, a direct implementation is accomplished using logic circuits.

- For each control line, one must find the Boolean expression in terms of the input to the control signal generator

Microprogrammed Control Unit

- Microprogramming was motivated by the desire to **reduce the complexities** involved with hardwired control.
- An **instruction** is implemented using a **set of micro-operations**.
- Associated with each **micro-operation** is a **set of control lines** that must be **activated** to carry out the corresponding microoperation.
- The idea of microprogrammed control is to **store** the **control signals** associated with the implementation of a certain instruction as a microprogram in a **special memory** called a control memory (CM).

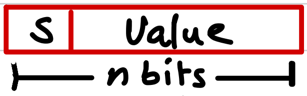
- A microprogram consists of a sequence of microinstructions.
  - A microinstruction is a **vector of bits**, where each bit is a control signal, condition code, or the address of the next microinstruction.
  - Microinstructions are fetched from CM the same way program instructions are fetched from main memory

- When an instruction is fetched from memory, the **op-code** field of the instruction will **determine which microprogram** is to be executed.

Chapter 10: Computer Arithmetic

⇒ Signed Integers :

① Signed Magnitude



Range : $-(2^{n-1}-1) \longrightarrow +(2^{n-1}-1)$

- two representations of zero (+0, −0)

② Biased

Bias $= 2^{n-1}-1$

Range : $-\text{Bias} \longrightarrow +(\text{Bias}+1)$

Binary to Decimal :

Binary value − Bias.

Decimal to Binary :

Decimal + Bias then convert to Binary.

③ 1's Complement

Range : $-(2^{n-1}-1) \longrightarrow +(2^{n-1}-1)$

- two representations of zero (e.g. +0, −0)

④ 2's Complement

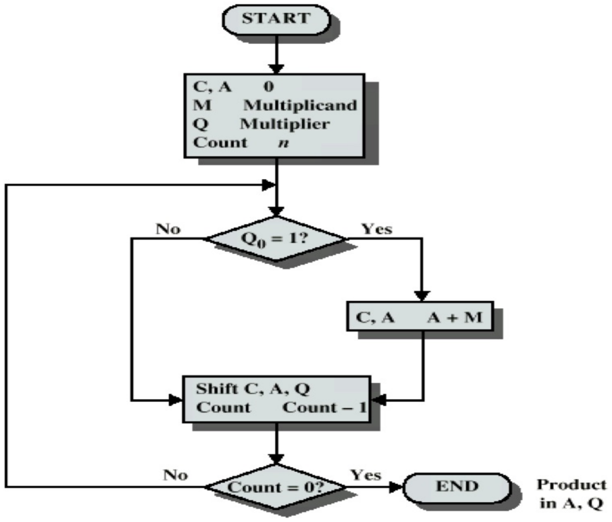
Range : $-2^{n-1} \longrightarrow +(2^{n-1}-1)$

⇒ Addition and Subtraction :

- when adding or Subtracting unsigned integers the Carry is important to indicates if the result is out of range or not.

- When adding or Subtracting signed integers the overflow is important to indicates if the result is out of range or not.

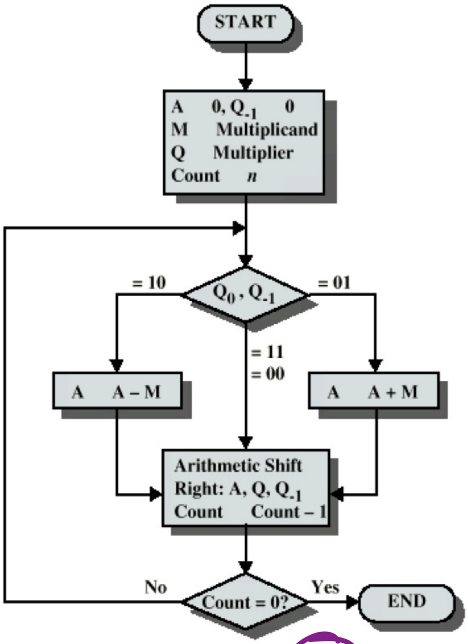
Flowchart for Unsigned Binary Multiplication



Execution of Example

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

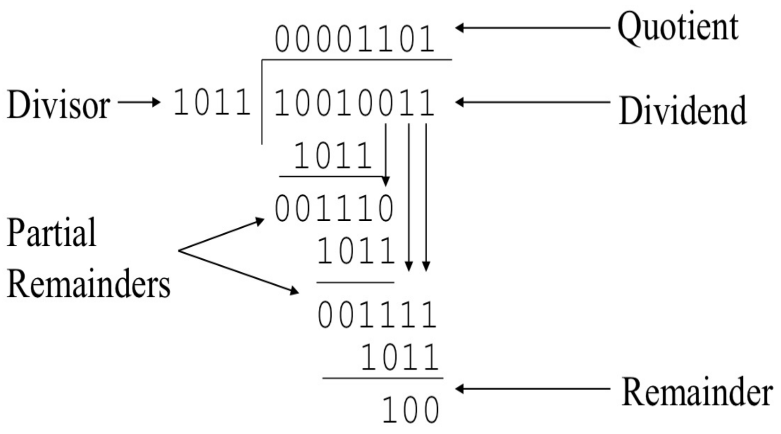
Booth's Algorithm



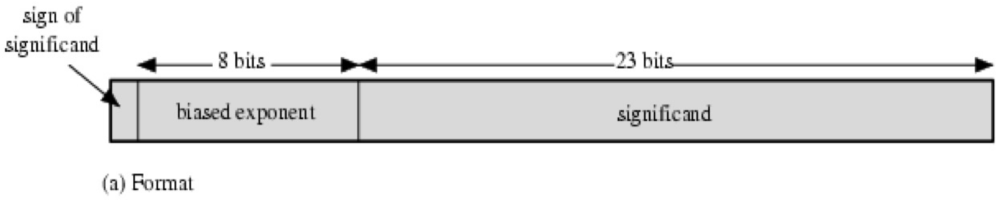
Example of Booth’s Algorithm

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

Division of Unsigned Binary Integers



Floating Point Examples



Converting from Floating Point

- E.g., What decimal value is represented by the following 32-bit floating point number?

$C17B0000_{16}$

Converting to Floating Point

- E.g., Express 36.5625_{10} as a 32-bit floating point number (in hexadecimal)