+ Chapter 3 (9th ed.) CPU Organization - Functions and Interconnections

STUDENTS-HUB.com

Program Concept

- Hardwired systems are inflexible
- General purpose hardware can do different tasks, given correct control signals
- Instead of re-wiring, supply a new set of control signals

What is a program?

- A sequence of steps
- For each step, an arithmetic or logical operation is done
- For each operation, a different set of control signals is needed

CPU Basics

- A typical CPU has three major components:
 - -Register Set,
 - The register set is usually a combination of generalpurpose and special-purpose registers.
 - General-purpose registers are used for any purpose.
 - Special-purpose registers have specific functions within the CPU.

-Arithmetic Logic Unit, and

-Control Unit (CU).

CPU Basics

- The Control Unit and the Arithmetic and Logic Unit constitute the Central Processing Unit
- Data and instructions need to get into the system and results out

—Input/output

 Temporary storage of code and results is needed

-Main memory

Function of Control Unit

 For each operation a unique code is provided

e.g. ADD, MOVE

- A hardware segment accepts the code and issues the control signals
- We have a computer!

CPU Basics



STUDENTS-HUB.com

Computer Components: Top Level View



STUDENTS-HUB Computer Components: Top-Level View

Registers

Memory buffer register (MBR)	•Contains a word to be stored in memory or sent to the I/O unit •Or is used to receive a word from memory or from the I/O unit
Memory address register (MAR)	•Specifies the address in memory of the word to be written from or read into the MBR
Instruction register (IR)	 holds the instruction currently being executed or decoded.
Program counter (PC)	•Contains the address of the next instruction pair to be fetched from memory
Accumulator (AC) and multiplier quotient (MQ)	•Employed to temporarily hold operands and results of ALU operations

STUDENTS-HUB.com

CPU Basics: Instruction Cycle

- A typical and simple execution cycle in a CPU is as follows:
 - The next instruction to be executed, whose address is obtained from the PC, is fetched from the memory and stored in the IR.
 - Instruction is decoded.
 - Operands are fetched from the memory and stored in CPU registers, if needed.
 - Instruction is executed.
 - Results are transferred from CPU registers to the memory, if needed.
- The execution cycle is repeated as long as there are more instructions to execute.
- A check for pending interrupts is usually included in the cycle.

Instruction Cycle

- Two steps:
 - -Fetch
 - -Execute



Figure 3.3 Basic Instruction Cycle

STUDENTS-HUB.com

Registers

- CPU must have some working space (temporary storage) Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy
- User Visible Registers
 - General Purpose
 - Data
 - Address
 - Condition Codes

General Purpose Registers

- Make them general purpose
 - Increase flexibility and programmer options
 - -Increase instruction size & complexity
- Make them specialized
 - -Smaller (faster) instructions
 - Less flexibility
- How Many GP Registers?
- How big?

- Memory Access Registers
 - Two registers are essential in memory write and read operations:
 - memory data register (MDR) and
 - memory address register (MAR).
 - The MDR and MAR are used exclusively by the CPU and are not directly accessible to programmers.
 - In order to perform a write operation into a specified memory location, the MDR and MAR are used as follows:
 - The word to be stored into the memory location is first loaded by the CPU into MDR
 - The address of the location into which the word is to be stored is loaded by the CPU into a MAR.

- Memory Access Registers
 - Similarly, to perform a memory read operation, the MDR and MAR are used as follows:
 - The address of the location from which the word is to be read is loaded into the MAR.
 - The required word will be loaded by the memory into the MDR ready for use by the CPU.
- Instruction Fetching Registers
 - Two main registers are involved in fetching an instruction for execution:
 - the program counter (PC) and
 - the instruction register (IR).

- Condition Registers
 - Condition registers, or flags, are used to maintain status information.
 - Some architectures contain a special Program Status Word (PSW) register.
 - The PSW contains bits that are set by the CPU to indicate the current status of an executing program.
 - These indicators are typically for arithmetic operations, interrupts, memory protection information, or processor status.

Sign of last result, Zero, Carry, Equal, Overflow, Interrupt enable/disable, Supervisor

- Special Purpose Address Registers
 - Index register
 - The index register holds an address displacement.
 - Index addressing is indicated in the instruction by including the name of the index register in parentheses and using the symbol X to indicate the constant to be added.
 - Segment pointers
 - The address issued by the processor should consist of a Segment Number (Base) and a Displacement (or an offset) within the segment.
 - A segment register holds the address of the base of the segment.

- Special Purpose Address Registers
 - Stack Pointer:
 - A stack is a data organization mechanism in which the last data item stored is the first data item retrieved.
 - Two specific operations can be performed on a stack. These are the Push and the Pop operations.
 - A specific register, called the stack pointer (SP), is used to indicate the stack location that can be addressed.
 - In the stack push operation, the SP value is used to indicate the location (called the top of the stack).
 - After storing (pushing) this value, the SP is incremented (In some architectures a SP is decremented (i.e. x86) as the stack grows low in memory).

- As long as there are instructions to execute, the next instruction is fetched from main memory.
- The instruction is executed based on the operation specified in the opcode field of the instruction.
- At the completion of the instruction execution, a test is made to determine whether an interrupt has occurred.
- An interrupt handling routine needs to be invoked in case of an interrupt.
- The basic actions during fetching an instruction, executing an instruction, or handling an interrupt are defined by a sequence of micro-operations.
- A group of control signals must be enabled in a prescribed sequence to trigger the execution of a micro-operation.

Fetch Instructions

- The sequence of events in fetching an instruction can be summarized as follows:
 - The contents of the PC are loaded into the MAR.
 - The value in the PC is incremented. (This operation can be done in parallel with a memory access).
 - As a result of a memory read operation, the instruction is loaded into the MDR.
 - The contents of the MDR are loaded into the IR.

Step	Micro-operation
+	MAR \leftarrow (PC): PC \leftarrow (PC) + 4
'0	
t ₁	MDR ← Mem[MAR]
t ₂	$IR \leftarrow (MDR)$

Execute Simple Arithmetic Operation

- Add R1, R2, R0
 - This instruction adds the contents of source registers R1 and R2, and stores the results in destination register R0. This addition can be executed as follows:
 - The registers R0, R1, R2, are extracted from the IR.
 - The contents of R1 and R2 are passed to the ALU for addition.
 - The output of the ALU is transferred to R0.

Step	Micro-operation
t _o	$R_0 \leftarrow (R_1) + (R_2)$

- Execute Simple Arithmetic Operation
 - Add X, R0
 - This instruction adds the contents of memory location X to register R0 and stores the result in R0. This addition can be executed as follows:
 - The memory location X is extracted from IR and loaded into MAR.
 - As a result of memory read operation, the contents of X are loaded into MDR.
 - The contents of MDR are added to the contents of R0.

Step	Micro-operation
to	MAR \leftarrow X
t ₁	MDR ← Mem[MAR]
t ₂	$R_0 \leftarrow (R_0) + (MDR)$

Format of Instructions and Data



(b) Integer format

Program Counter (PC) = Address of instruction Instruction Register (IR) = Instruction being executed Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory 0010 = Store AC to Memory 0101 = Add to AC from Memory

(d) Partial list of opcodes

STUDENTS-HUB.com

Example of Program Execution



Example of Program Execution

- 1. PC contains address of first instruction: 300h
- 2. PC value moved to AR or MAR
- Instruction 1940h (hex) is loaded into IR (IR ← M[AR])
- 4. Increment PC.
- 5. The "1" value in IR means LOAD AC.
- 6. The "940" value is the memory address for data to be read and loaded to AC: AC ← M[AR]
- 7. AC gets the value of 3h. Increment PC
- 8. Next inst. is 5941h: the value "5" means add to AC from memory location "941". Increment PC
- 9. Next inst. is 2941h: the value "2" means store AC to memory location "941"

Instruction Cycle State Diagram



Execution cycle may reference memory more than once. The operation could be an I/O type. For some instruction, some states may be null and others may be accessed more than once. STUDENTS-HUB.com Uploaded By: anonymous

An Abstract (Simplified) View:



- elements that contain states (sequential)

Interrupts

 Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing. *Improves process efficiency*.

Classes:

- Program (condition occurs as a result of instruction execution)
 —e.g. Arithmetic overflow, division by zero
- Timer
 - Generated by internal processor timerUsed in pre-emptive multi-tasking
- I/O (from I/O controller) —to signal normal completion or error
- Hardware failure

-e.g. memory parity error, power failure STUDENTS-HUB.com

Program Flow Control



Program Flow Control



- 1. User program executes codes 1, 2, and 3 that do not involve the I/O.
- 2. It interleaves the codes with WRITE calls to an I/O program.
- The IO program has a sequence of instructions, 4, to prep for the I/O operation. It has the actual IO command, and a sequence of instructions, 5, to complete the operation (i.e. set flag for success or failure).
- 4. Since no interrupt, the IO command may take long, and the program has to wait for the IO device to perform what it was asked to do.
- 5. In this case, the IO program is hung up waiting, and the user program is stopped at WRITE call location.

(a) No interrupts STUDENTS-HUB.com

Program Flow Control



With interrupts, Processor can execute other inst. While an IO operation is in progress. A WRITE call is made and the IO preparation code,4, and the I/O command are executed.

- . User program resumes execution while the external device is busy doing what it was told to do via the I/O command (print data).
- . When external device is ready to be serviced again, its I/O module sends an Interrupt Request signal to the processor.
- Processor suspends operations to handle the Interrupt (point X) (interrupt handler). Normal operation then resumes.
- 6. Interrupt handling code is not part of the user code.
 - . Interrupt handling is the responsibility of the Processor and OS.

(b) Interrupts; short I/O wait STUDENTS-HUB.com

Interrupt Cycle

- Added to instruction cycle
- Processor checks for interrupt

-Indicated by an interrupt signal

- If no interrupt, fetch next instruction
- If interrupt pending:
 - -Suspend execution of current program
 - -Save context
 - Set PC to start address of interrupt handler routine
 - —Process interrupt
 - Restore context and continue interrupted program

STUDENTS-HUB.com

Transfer of Control via Interrupts



Instruction Cycle with Interrupts



If Interrupt is pending:

-Suspend execution of current program and save address of next instruction.

- Set the PC to the starting address of an Interrupt Handler Routine and then fetch the first instruction in the handler program

STUDENTS-HUB.com

Instruction Cycle (with Interrupts) -State Diagram



STUDENTS-HUB.com

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
- 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



Multiple Interrupts – Nested



STUDENTS-HUB.com

Uploaded By: anonymous

Time Sequence of Multiple Interrupts



STUDENTS-HUB.com

Uploaded By: anonymous

Connecting

- All the units must be connected
- Different type of connection for different type of unit
 - -Memory
 - —Input/Output

-CPU







CPU Connection

- Reads instruction and data
- Writes out data (after processing)
- Sends control signals to other units
- Receives (& acts on) interrupts

Memory Connection

- Receives and sends data
- Receives addresses (of locations)
- Receives control signals
 - -Read
 - -Write
 - -Timing

Input/Output Connection

- Similar to memory from computer's viewpoint
 - -Output
 - Receive data from computer
 - Send data to peripheral
 - —Input
 - Receive data from peripheral
 - Send data to computer
- Receive control signals from computer
- Send control signals to peripherals —e.g. spin disk
- Receive addresses from computer
 - -e.g. port number to identify peripheral
- Send interrupt signals (control)

Buses

- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP)

What is a Bus?

- A communication pathway connecting two or more devices
- Usually broadcast
- Often grouped
 - —A number of channels in one bus
 - —e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown

Computer System Buses

- What do buses look like?
 - -Parallel lines on circuit boards
 - -Ribbon cables
 - -Strip connectors on mother boards
 - -Sets of wires





STUDENTS-HUB.com

Bus Types

- Dedicated
 - -Separate data & address lines

Multiplexed

- -Shared lines
- -Address valid or data valid control line
- -Advantage fewer lines
- -Disadvantages
 - More complex control
 - Ultimate performance

Data Bus

Carries data

—Remember that there is no difference between "data" and "instruction" at this level

Width is a key determinant of performance

-8, 16, 32, 64 bit

Address bus

- Identify the source or destination of data
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
 - —e.g. 8080 has 16 bit address bus giving 64k address space

Control Bus

- Control and timing information
 - —Memory read/write signal
 - —Interrupt request
 - -Clock signals

Bus Interconnection Scheme



Uploaded By: anonymous

CPU local (internal) Bus Organization

One-Bus Organization

- -Using one bus, the CPU registers and the ALU use a single bus to move outgoing and incoming data.
- —Since a bus can handle only a single data movement within one clock cycle, twooperand operations will need two cycles to fetch the operands
- —Additional registers may be needed to buffer data for the ALU
- -This bus organization is the **simplest** and **least expensive**, but it **limits** the amount of **data transfer** that can be done in the same clock cycle, which will **slow down** the overall **performance**.

Single Bus Problems

- Lots of devices on one bus leads to:
 - -Propagation delays
 - Long data paths mean that co-ordination of bus use can adversely affect performance
 - If aggregate data transfer approaches bus capacity
- Most systems use multiple buses to overcome these problems

One-Bus Organization



Two-Bus Organization

- Using two buses is a faster solution than the one-bus organization.
- In this case, general-purpose registers are connected to both buses.
- Data can be transferred from two different registers to the ALU at the same time.
- —Therefore, a two operand operation can fetch both operands in 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 the two operands.
- In some cases, one of the buses may be dedicated for moving data into registers (in-bus), while the other is dedicated for transferring data out of the registers (out-bus).

STUDENTS-HUB.com

Uploaded By: anonymous

Two-Bus Organization



Two-Bus Datapath

STUDENTS-HUB.com

Uploaded By: anonymous

Three-Bus Organization

- —In a three-bus organization, two buses may be used as source buses while the third is used as destination.
- —The source buses move data out of registers (out-bus), and the destination bus may 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.
- -The **more buses** we have, the **more data** we can move within a single clock cycle.

 However, increasing the number of buses will also increase the complexity of the hardware. STUDENTS-HUB.com

Three-Bus Organization



Three-Bus Datapath

Bus Arbitration

- More than one module controlling the bus —e.g. CPU and DMA controller
- Only one module may control bus at one time
- Arbitration may be centralised or distributed
 - -Centralised : Only one module (bus controller or arbiter) may control bus at one time
 - -Distributed : More than one module controlling the bus
 - e.g. CPU and DMA controller

Timing

- Co-ordination of events on bus
- Bus use either synchronous or asynchronous timing.

Synchronous

- -Events determined by clock signals
- -Control Bus includes *clock line*
- —A single 1-0 transition is referred to as is a bus cycle or clock cycle
- -All devices on the bus can read clock line
- -Usually sync on leading edge
- -Usually a single cycle for an *event*

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.

STUDENTS-HUB.com

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.



Control Unit

- 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
- Let us explain the implementation using simple example.

Hardwired Implementation example

- Assume that the instruction set of a machine has the three instructions: Instx, Inst-y, and Inst-z;
- and A, B, C, D, E, F, G, and H are control lines.
- The following table shows the control lines that should be activated for the three instructions at the three steps t0, t1, and t2.

Step	Inst-x	Inst-y	Inst-z
t ₀	D, B, E	F, H, G	Е, Н
t_1	С, А, Н	G	D, A, C
t_2	G, C	В, С	

Hardwired Implementation example

Step	Inst-x	Inst-y	Inst-z			
t ₀	D, B, E	F, H, G	Е, Н			
t_1	С, А, Н	G	D, A, C			
t_2	G, C	В, С				

The Boolean expressions for control lines A, B, and C can be obtained as follows:

$$A = \text{Inst-x} \cdot t_1 + \text{Inst-z} \cdot t_1 = (\text{Inst-x} + \text{Inst-z}) \cdot t_1$$

$$B = \text{Inst-x} \cdot t_0 + \text{Inst-y} \cdot t_2$$

$$C = \text{Inst-x} \cdot t_1 + \text{Inst-x} \cdot t_2 + \text{Inst-y} \cdot t_2 + \text{Inst-z} \cdot t_1$$

$$= (\text{Inst-x} + \text{Inst-z}) \cdot t_1 + (\text{Inst-x} + \text{Inst-y}) \cdot t_2$$

Boolean expressions for the rest of the control lines can be Obtained in a similar way. Uploaded By: anonymous

Hardwired Implementation example

The Boolean expression for control lines A, B and C

$$A = \text{Inst-x} \cdot t_1 + \text{Inst-z} \cdot t_1 = (\text{Inst-x} + \text{Inst-z}) \cdot t_1$$

$$B = \text{Inst-x} \cdot t_0 + \text{Inst-y} \cdot t_2$$

$$C = \text{Inst-x} \cdot t_1 + \text{Inst-x} \cdot t_2 + \text{Inst-y} \cdot t_2 + \text{Inst-z} \cdot t_1$$

$$= (\text{Inst-x} + \text{Inst-z}) \cdot t_1 + (\text{Inst-x} + \text{Inst-y}) \cdot t_2$$

Logic Circuit for control lines A, B and C



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).

STUDENTS-HUB.com

Uploaded By: anonymous

Microprogrammed Control Unit

- 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.

Example#1

A memory, shown below, is used to store instructions and data for a basic machine with three 16-bit registers, namely, **R0**, **R1**, and **R2**. Moreover, this machine has **PC** and **IR** registers. Assume that the first instruction of a given program is stored at memory address location **100H**. All numbers are in Hexadecimal format.

Opcode (2bits)	Dest. Reg (2bits)	Memory Address (12bits)				
	Instruction format					
2's complement Signed integers (16 bits)						

Data format

Opcode 00 is to load register Ri from the specified memory address

Opcode 01 is to store register Ri to the memory at the specified address

Opcode 10 is to add the content of register Ri to the content of the specified memory location and store the result in Ri

Opcode 11 is to subtract the memory content at the specified memory address from the content of register *Ri and store the result in Ri*

Memory	100	101	102	103	104	105	106	107	108
Address									
Content	1106	2107	9108	E108	5108	0001	0004	0005	0003

Example#1

 a) Fill in the table above with the values for PC, IR, R0, R1 and R2 registers? Initial values mean the values of the registers prior to fetching the first instruction from memory.

	PC	IR	R0	R1	R2
Registers Initial Values	100H	0000	0000	0000	0000
After 1 st instruction execution	101H	1106	0000	0004	0000
After 2 nd instruction execution	102H	2107	0000	0004	0005
After 3 rd instruction execution	103H	9108	0000	0007	0005
After 4 th instruction execution	104H	E108	0000	0007	0002
After 5 th instruction execution	104H	5108	0000	0007	0002

• b) What is the range of the signed integer values that can be stored in this memory?

The data has 16-bit width

$$-2^{n-1} \rightarrow 2^{n-1} - 1 \rightarrow -2^{15} \rightarrow 2^{15} - 1$$

STUDENTS-HUB.com
Example#1

• c) write the machine code for the instruction ADD R0, [105H].

```
ADD has opcode of 10
```

```
Ro is represented by 00
```

Thus, the machine code of is as follows:

```
(1000 0001 0000 0101) 2
```

(8105)16

d) Using the available above four instructions, write an assembly code to perform Mem[105]=Mem[106]+Mem[107]

LOAD R0, MEM [106] // load memory location 106 to register R0

ADD R0, MEM [107] // Add the content of memory location 107 to the

//content of register R0 and store the result in R0

STORE R0, MEM [105] // Store the content of R0 to memory location 105