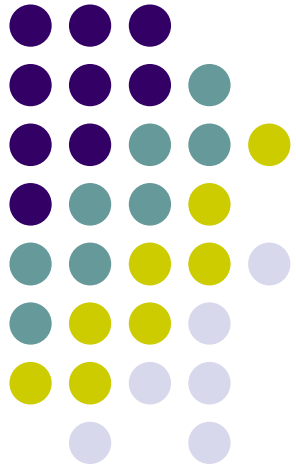
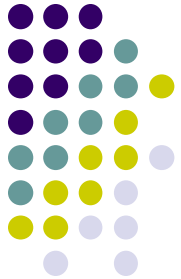
Chapter 7. Basic Processing Unit

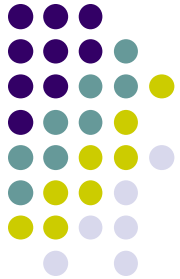


Instruction Set Processor (ISP)



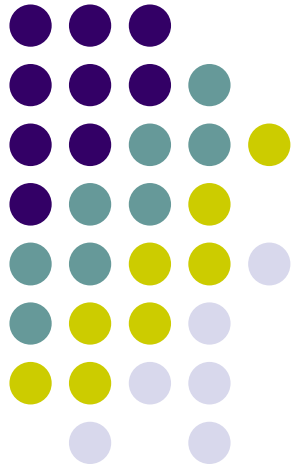
- Processing unit, which executes machine instructions and coordinates the activities of other units is often called the **Instruction Set Processor (ISP)**, or simply the **processor.**

Overview

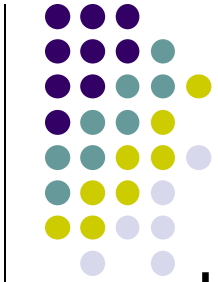


- Instruction Set Processor (ISP)
- Central Processing Unit (CPU)
- A typical computing task consists of a series of steps specified by a sequence of machine instructions that constitute a program (see chapter 2)
  - **A Program = A Sequence of Machine Instructions**
  - **Each Instruction = A Sequence of MicroInstructions (very basic operations)**
- **An instruction is executed by carrying out a sequence of more rudimentary (basic) operations.**
  - Example: An Instruction: **Add (R3), R1** requires seven microinstructions (basic steps) involving around 25 control signal.

Some Fundamental Concepts



Fundamental Concepts



- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the **address of memory location that contains the next instruction** to be executed using the **Program Counter register (PC).**
- **Instruction Register (IR)** holds the instruction that is currently being decoded and executed by generating the **sequence of control signals corresponding to this current instruction by the ***Control Unit*** of the CPU.**

Executing an Instruction: Fetch & Execute

- Fetch the next instruction by fetching the contents of the memory location whose address is in the PC. The contents of this location are loaded into the IR **(this is fetch phase of the instruction).**

$$IR \leftarrow [\,[PC]\,]$$

- Assuming that the memory is byte addressable and each instruction occupies exactly 4 bytes in memory, increment the contents of the PC by 4 **(also part of the fetch phase).**

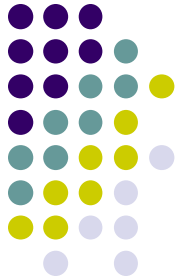
$$PC \leftarrow [PC] + 4$$

- Carry out the actions specified by the instruction in the IR **(execution phase).** This requires generating a number of **control signals** in an appropriate order.

Internal organization of a processor

- Recall that a processor has several registers/building blocks:
  - Memory address register (MAR)
  - Memory data register (MDR)
  - Program Counter (PC)
  - Instruction Register (IR)
  - General purpose registers R0 - R(n-1)
  - Arithmetic and logic unit (ALU)
  - Control unit.
- How are these units organized and how do they communicate with each other?

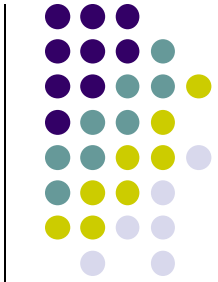
CPU Organization: Single Bus Datapath



MDR HAS TWO INPUTS AND TWO OUTPUTS

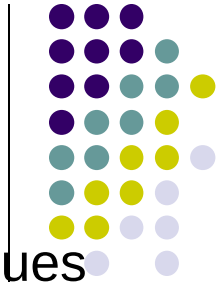
Datapath

Single bus organization



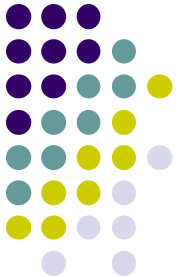
- Single bus organization:
  - ALU, control unit and all the registers are connected via a single common bus.
  - Bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- Data lines of the external memory bus are connected to the internal processor bus via MDR.
  - Register MDR has two inputs and two outputs.
  - Data may be loaded to (from) MDR from (to) internal processor bus or external memory bus.
- Address lines of the external memory bus are connected to the internal processor bus via MAR.
  - MAR receives input from the internal processor bus.
  - MAR provides output to external memory bus.

Single bus organization (contd..)



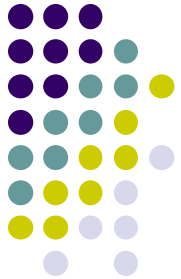
- Instruction decoder and control logic block, or control unit issues signals to control the operation of all units inside the processor and for interacting with the memory bus.
  - Control signals depend on the instruction loaded in the Instruction Register (IR)
- Outputs from the control logic block are connected to:
  - Control lines of the memory bus.
  - ALU, to determine which operation is to be performed.
  - Select input of the multiplexer MUX to select between Register Y and constant 4.
  - Control lines of the registers, to select the registers.

Single bus organization (contd..)



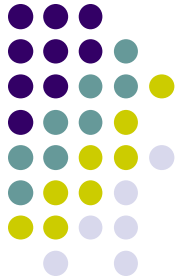
- Registers Y, Z, and TEMP:
  - Used by the processor for temporary storage during execution of some instructions.
  - Note that Registers R0 to R(n-1) are used to store data generated by one instruction for later use by another instruction.
  - Data is stored in R0 through R(n-1) after the execution of an instruction.
- Multiplexer MUX selects either the output of register Y or a constant 4, depending upon the control input Select.
  - Constant 4 is used to increment the value of the PC.

Registers and the bus (contd..)

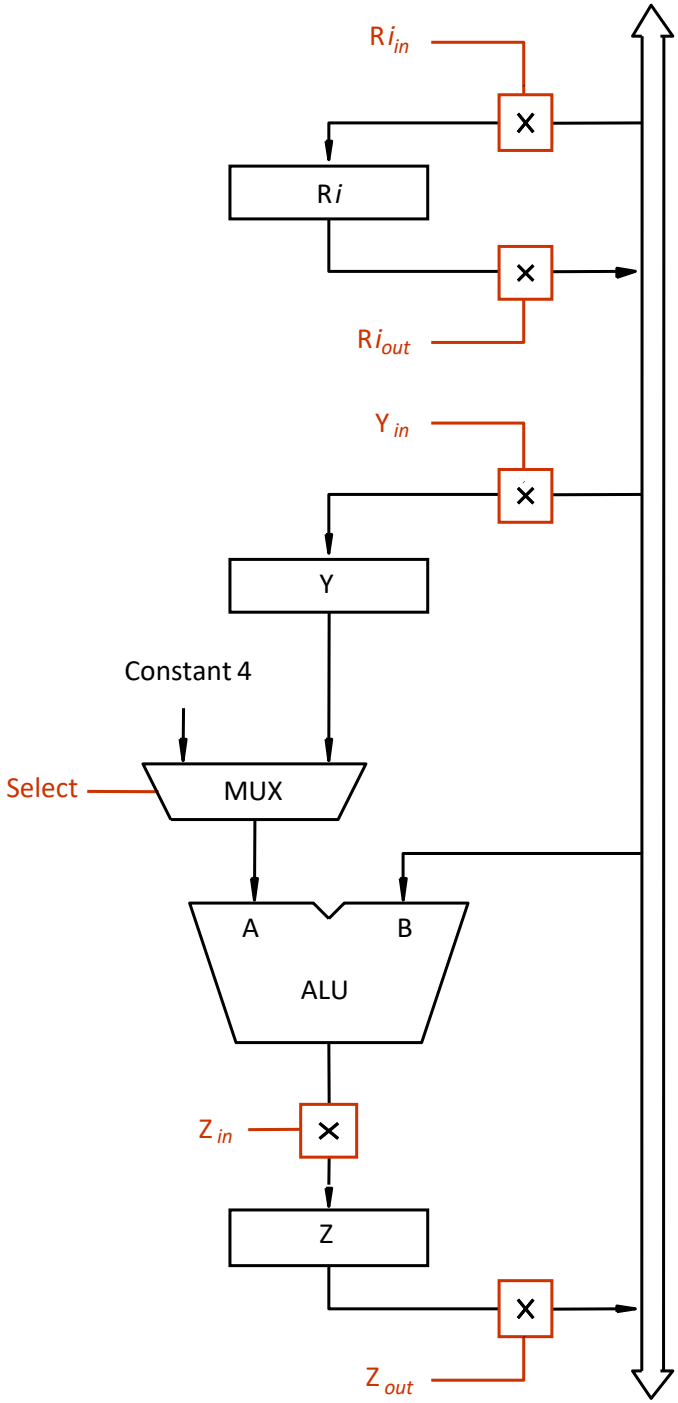


- A bus may be viewed as a collection of parallel wires.
- Buses have no memory:
  - They are just a collection of wires.
- When data is on the bus, all registers can "see" that data at their inputs.
- A register may place its contents onto the bus.

Registers and the bus (contd..)



- At any one time, only one register may output its contents to the bus:
  - Which register outputs its content to the bus is determined by the control signal issued by the control logic.
  - Control signal depends on the instruction loaded in the instruction register IR.
- Registers can load data from the bus:
  - Which registers load data from the bus is determined by the control signal issued by the control logic.
- Registers are clocked (sequential) entities (unlike ALU which is purely combinatorial).



Registers are connected to the bus via switches controlled by the signals Rin & Rout.

Each register Ri has two control signals, Ri_{in} and Ri_{out}.

If $Ri_{in}=1$, the data from the bus is loaded into the register.

If $Ri_{out}=1$, the data from the register is loaded onto the bus.

The same holds for registers Y and Z as well.

Figure 7.2. Input and output gating for the registers in Figure 7.1.

Input and output gating for one register bit [SELF STUDY]

- All operations and data transfers are controlled by the processor clock.
- When Ri_{out}= 1, the output of the D-Flip Flop is Placed on the Bus (i.e., register data is **PUT** on the CPU datapath)
- When Ri_{in}= 0, the output of the D-Flip Flop is connected to its input (So, the D-FF **HOLDS** its contents)
- When Ri_{in}= 1, the input of the D-Flip Flop is connected with the Bus (So, FF **LOADS** data from the Bus on next clock pulse)

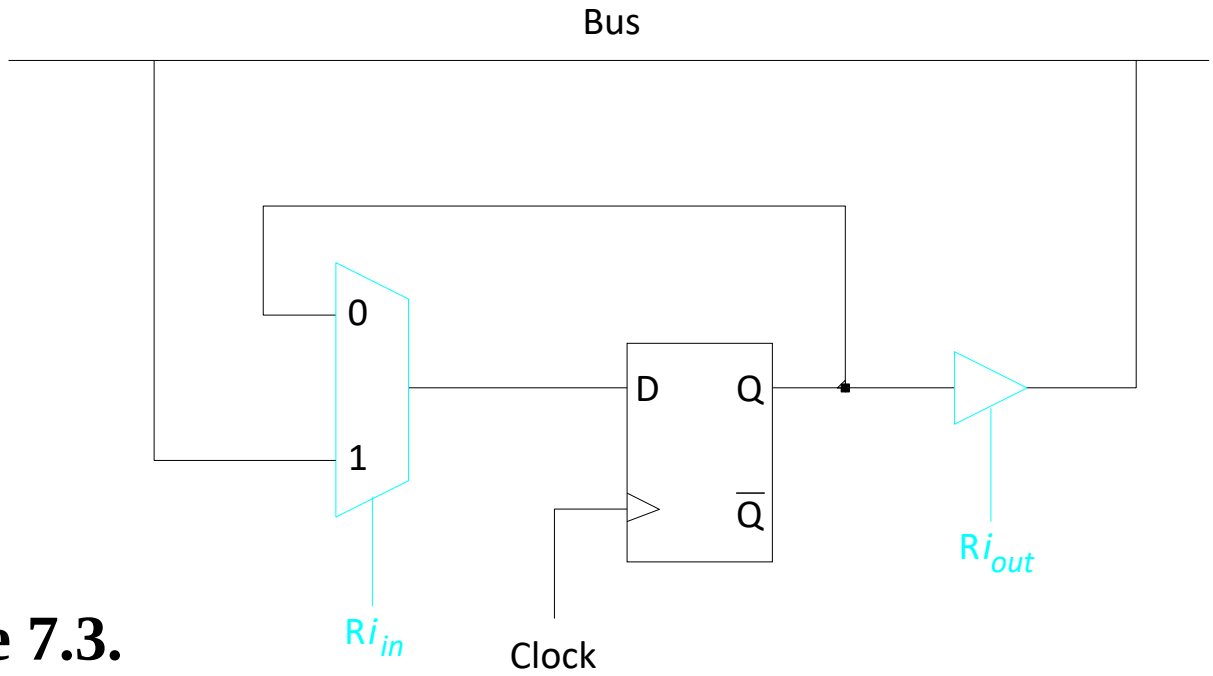
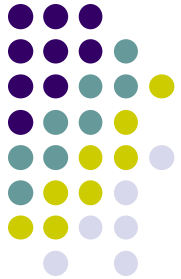


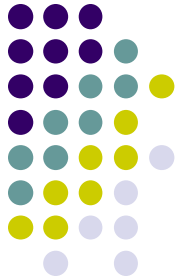
Figure 7.3.

Executing an Instruction: Examples



- *Transfer a word of data from one processor register to another OR to the ALU* (i.e., by placing the register content onto the internal processor bus – see in the next slide fig).
- *Perform an arithmetic (e.g., ADD, SUB, MUL) or a logic operation* (e.g., XOR, SHIFT, ROTATE) and store the result in a processor register.
- *Fetch the contents of a given memory location and load them into a processor register*. **(LOAD)**
- *Store a word of data from a processor register into a given memory location*. **(STORE)**

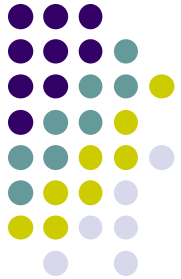
Performing an Arithmetic or Logic Operation



- The ALU is a combinational circuit that has no internal storage. So, Registers are needed !!!
- **ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.**
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

Add R_1, R_2, R_3

Performing an arithmetic operation



Add the contents of registers *R1* and *R2* and place the result in *R3*.
That is: *R3 = R1 + R2*

1. Place the contents of register R1 into the Y register in the first clock cycle.

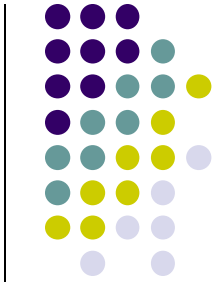
2. Place the contents of register R2 onto the bus in the second clock cycle.

Both inputs to the ALU are now valid. Select register Y, and assert the ALU command F=A+B.

3. In the third clock cycle, Z register has latched the output of the ALU. Thus the contents of the Z register can be copied into register R3.

1. **R1out, Yin**
2. **R2out, SelectY, Add, Zin**
3. **Zout, R3in**

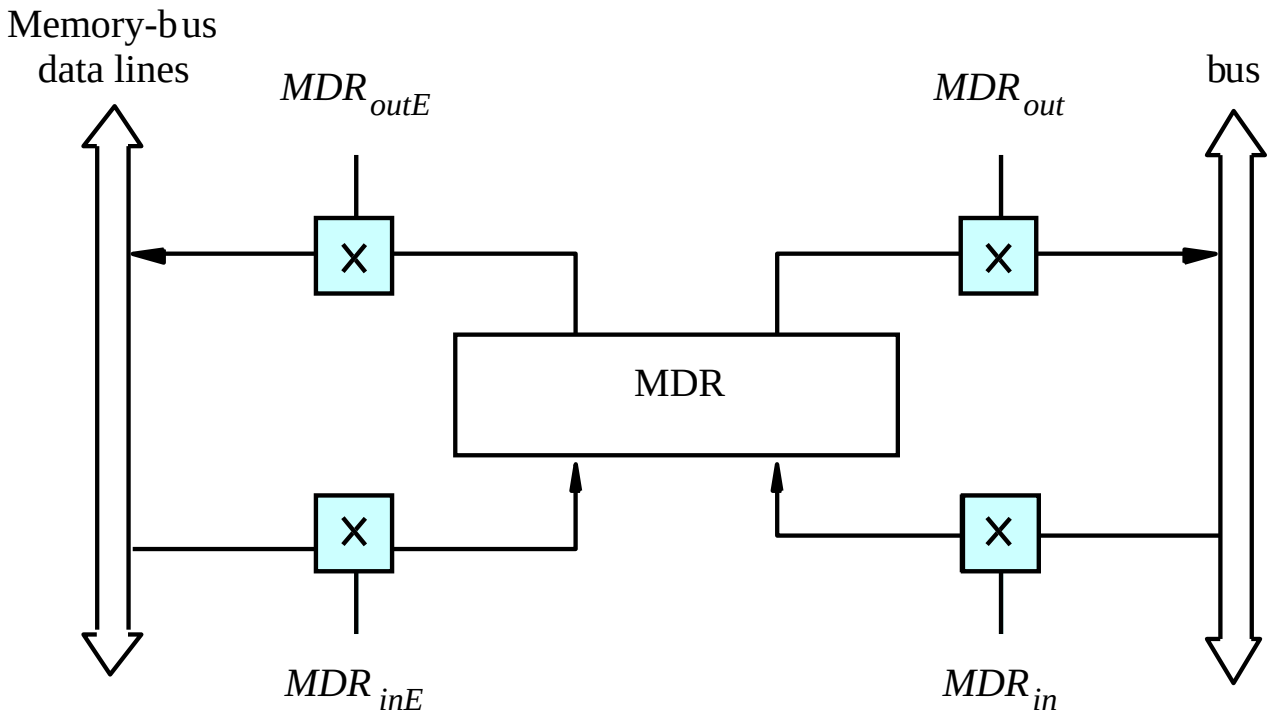
Fetching a word from memory



- Processor has to specify the address of the memory location where this information is stored and request a Read operation.
- Processor transfers the required address to *MAR*.
  - Output of *MAR* is connected to the address lines of the memory bus.
- Processor uses the control lines of the memory bus to indicate that a *Read* operation is needed.
- Requested information are received from the memory and are stored in *MDR*.
  - Transferred from *MDR* to other registers.

Fetching a word from memory (contd..)

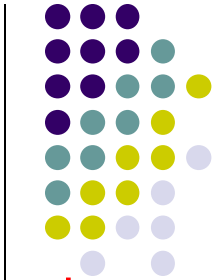
Connections for register MDR



MDR_{outE} and MDR_{inE} control connection to external bus.

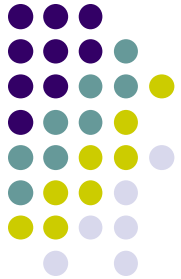
MDR_{out} and MDR_{in} control connection to internal bus.

Fetching a word from memory (contd..)



- Timing of the internal processor operations must be coordinated with the response time of memory Read operations.
- Processor completes one internal data transfer in one clock cycle.
- Memory response time for a Read operation is variable and usually longer than one clock cycle.
  - Processor waits until it receives an indication that the requested Read has been completed.
  - Control signal Memory Function Completed (MFC) is used for this purpose.
  - **MFC** is set to **1** by the memory to indicate that the contents of the specified location have been read and are **available** on the data lines of the memory **bus**.

Fetching a word from memory (contd..)



MOVE (R1), R2

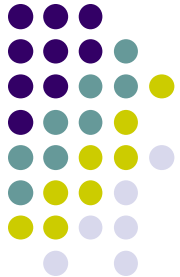
1. Load the contents of Register *R1* into *MAR*.
2. Start a *Read* operation on the memory bus.
3. Wait for *MFC* response from the memory.
4. Load *MDR* from the memory bus.
5. Load the contents of *MDR* into Register *R2*.

Steps can be performed separately, some may be combined.

1. Steps 1 and 2 can be combined.
   - Load ***R1*** to ***MAR*** and activate ***Read*** control signal simultaneously.
2. Steps 3 and 4 can be combined.
   - Activate control signal MDR_{inE} while waiting for response from the memory MFC.
3. Last step loads the contents of ***MDR*** into Register ***R2***.

Hence, Memory ***Read*** operation takes 3 steps.

Fetching a word from memory (contd..)



MOVE (R1) , R2: Memory operation takes 3 steps.

Step 1:

- Place *R1* onto the internal processor bus.
- Load the contents of the bus into *MAR*.
- Activate the *Read* control signal.
- $R1_{out}$, MAR_{in}, ***Read.***

Step 2:

- Wait for MFC from the memory.
- Activate the control signal to load data from external bus to *MDR*.
- MDR_{inE}, ***WMFC***

Step 3:

- Place the contents of *MDR* onto the internal processor bus.
- Load the contents of the bus into Register *R2*.
- MDR_{outI}, $R2_{in}$

Storing a word into memory

MOVE R2, (R1): **Memory operation takes 3 steps.**

Step 1:

- Place *R1* onto the internal processor bus.
- Load the contents of the internal processor bus into *MAR*.
- $R1_{out}$, MAR_{in}.

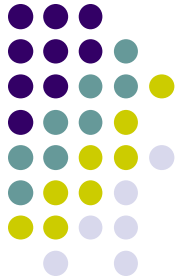
Step 2:

- Place *R2* onto the internal processor bus.
- Load the contents of the internal processor bus into *MDR*.
- Activate Write operation.
- $R2_{out}$, MDR_{in}, *Write*

Step 3:

- Place the contents of *MDR* into the external memory bus.
- Wait for the memory write operation to be completed MFC.
- MDR_{outE}, *WMFC*

Execution of a complete instruction



Add the contents of a memory location pointed to by Register *R3* to register *R1*.

ADD (R3), R1

To execute the instruction we must execute the following tasks:

1. Fetch the instruction.
2. Fetch the operand (contents of the memory location pointed to by *R3*.)
3. Perform the addition.
4. Load the result into *R1*.

Execution of a complete instruction

Task 1: Fetch the instruction

Recall that:

- *PC* holds the address of the memory location which has the next instruction to be executed.
- *IR* holds the instruction currently being executed.

Step 1

- Load the contents of *PC* to *MAR*.
- Activate the *Read* control signal.
- Increment the contents of the *PC* by 4.
- $\boldsymbol{PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}}$.

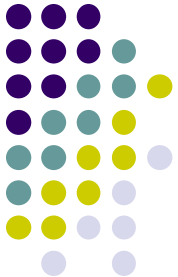
Step 2

- Update the contents of the *PC*.
- Copy the updated *PC* to Register *Y* (useful for Branch instructions).
- Activate the control signal to load data from external bus to *MDR*
- Wait for *MFC* from memory.
- $\boldsymbol{Z_{out}, PC_{in}, Y_{in}, MDR_{inE}, WMFC}$

Step 3

- Place the contents of *MDR* onto the bus.
- Load the *IR* with the contents of the bus.
- $\boldsymbol{MDR_{out}, IR_{in}}$

Execution of a complete instruction (contd..)



Task 2. Fetch the operand (contents of memory pointed to by *R3*.)
Task 3. Perform the addition.
Task 4. Load the result into *R1*.

Step 4: - Place the contents of Register *R3* onto internal processor bus.
- Load the contents of the bus onto *MAR*.
- Activate the *Read* control signal.
- $\boldsymbol{R3_{out}, MAR_{in}, Read}$

Step 5: - Place the contents of *R1* onto the bus.
- Load the contents of the bus into Register *Y (*Recall one operand in *Y)*.
- Wait for *MFC*.
- $\boldsymbol{R1_{out}, Y_{in}, MDR_{inE}, WMFC}$

Step 6: - Load the contents of *MDR* onto the internal processor bus.
- Select *Y,* and perform the addition.
- Place the result in *Z*.
- $\boldsymbol{MDR_{out}}$**, SelectY, Add,** $\boldsymbol{Z_{in}}$.

Step 7: - Place the contents of Register *Z* onto the internal processor bus.
- Place the contents of the bus into Register *R1*.
- $\boldsymbol{Z_{out}, R1_{in}}$

Execution of a Complete Instruction

Add (R3), R1

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

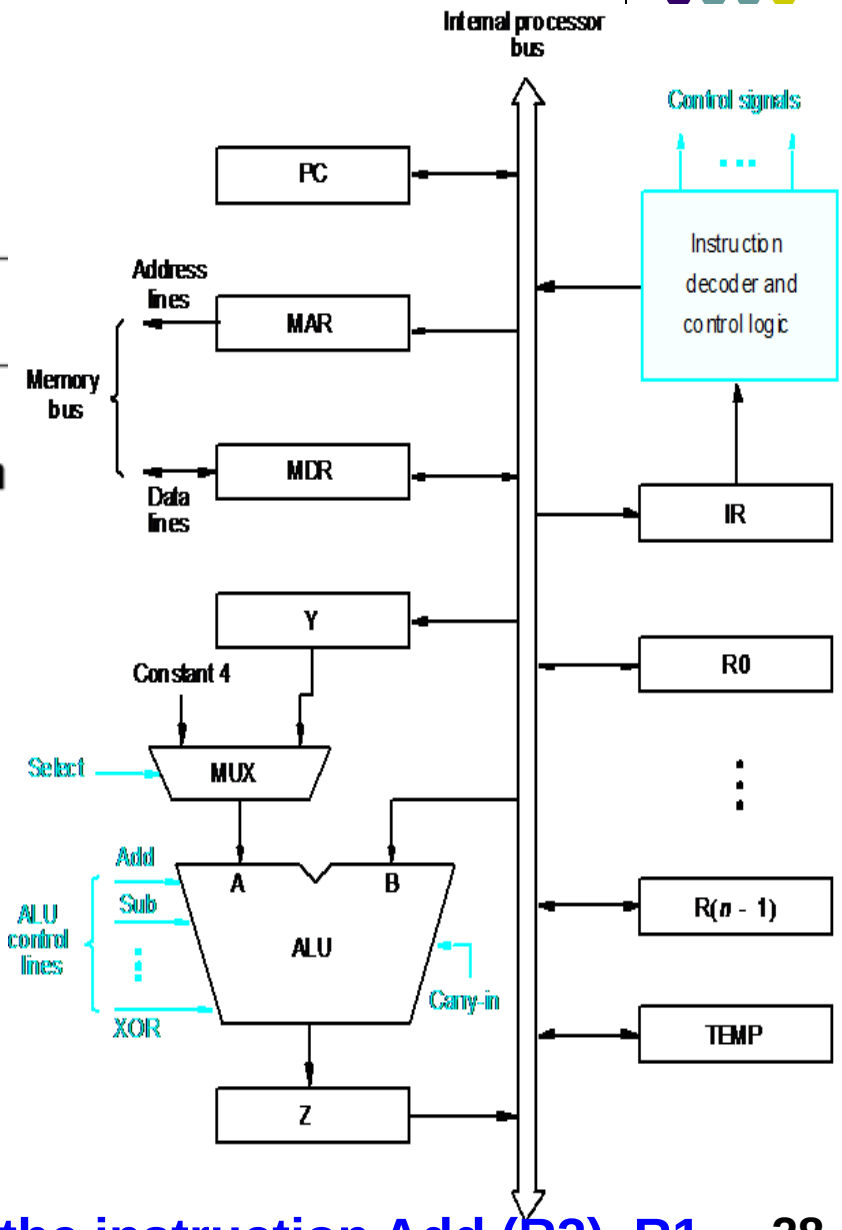
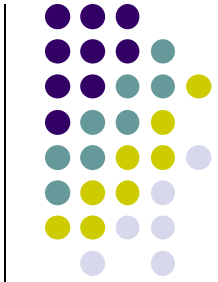


Fig 7.6: Control Sequence for executing the instruction Add (R3), R1

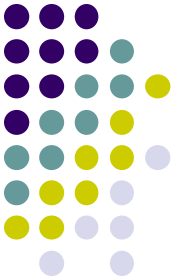
Execution of Branch Instructions (Self Study)



- **JMP LABEL; BRANCH LABEL** (unconditional branch)
- A branch instruction **replaces the contents of PC with the branch target address, which is usually obtained by adding an offset LABEL given in the branch instruction. (Offset-field-of-Instruction)**
- Thus, the offset LABEL is the difference between the branch target address and the address of next (immediately following) instruction (because, when current instruction is running, already PC is loaded with PC+4, i.e., the address of the next instruction)
- Conditional branch depends on the processor's status bits/flags/condition codes.

(JE/JNE/JZ) label; (Branch<0 / Branch==0 /Branch>0) label

Execution of Unconditional Branch Instructions

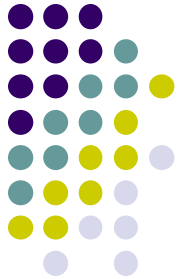


Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMF C
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Select Y, Add, Z_{in},
5	Z_{out}, PC_{in}, End

Figure 7.7. Control sequence for an unconditional branch instruction.

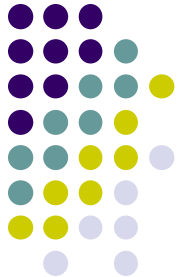
Note in Step 4 – the ADD adds the contents of Y (which now holds the value of the current PC (see the Yin in the step 2)) with the Offset-field-of-IR (which contains the offset to be added to PC to make PC point to the branch target address)

Multiple-bus organization



- Simple single-bus structure:
  - Results in long control sequences, because only one data item can be transferred over the bus in a clock cycle.
- Most commercial processors provide multiple internal paths to enable several transfers to take place in parallel.
  - Multiple-bus organization.

Multiple bus organization (contd..)

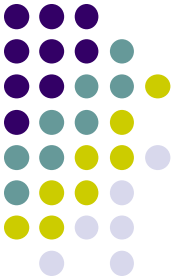


- Three-bus organization to connect the registers and the ALU of a processor.
- All general-purpose registers are combined into a single block called register file.
  - Register file has three ports.
  - Two outputs ports connected to buses A and B, allowing the contents of two different registers to be accessed simultaneously, and placed on buses A and B.
  - Third input port allows the data on bus C to be loaded into a third register during the same clock cycle.
- Inputs to the ALU and outputs from the ALU:
  - Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.
  - Result is transferred to the destination over bus C.

Multiple bus organization (contd..)

- ALU can also pass one of its two input operands unmodified if needed:
  - Control signals for such an operation are $R=A$ or $R=B$.
- Three bus arrangement obviates the need for Registers Y and Z in the single bus organization.
- Incrementer unit:
  - Used to increment the PC by 4.
  - Source for the constant 4 at the ALU multiplexer can be used to increment other addresses such as the memory addresses in multiple load/store instructions.

Multiple-Bus Organization: Three Bus CPU Datapath Architecture



Summary:

GPR ← bus C
GPR → bus A, B

ALU ← bus A OR 4, bus B
ALU → bus C

MDR, PC, IR ← bus C
MDR → bus A, B
PC → bus B *ONLY*
IR → bus A *ONLY*

MAR ← bus C

GPR means "General Purpose Registers"

→ Means OUTPUT
← Means INPUT

R=A means Transfer the Contents of bus A to bus C

R=B means Transfer the Contents of bus B to bus C

Control Sequence in Three-Bus Architecture

Add R4, R5, R6

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$,

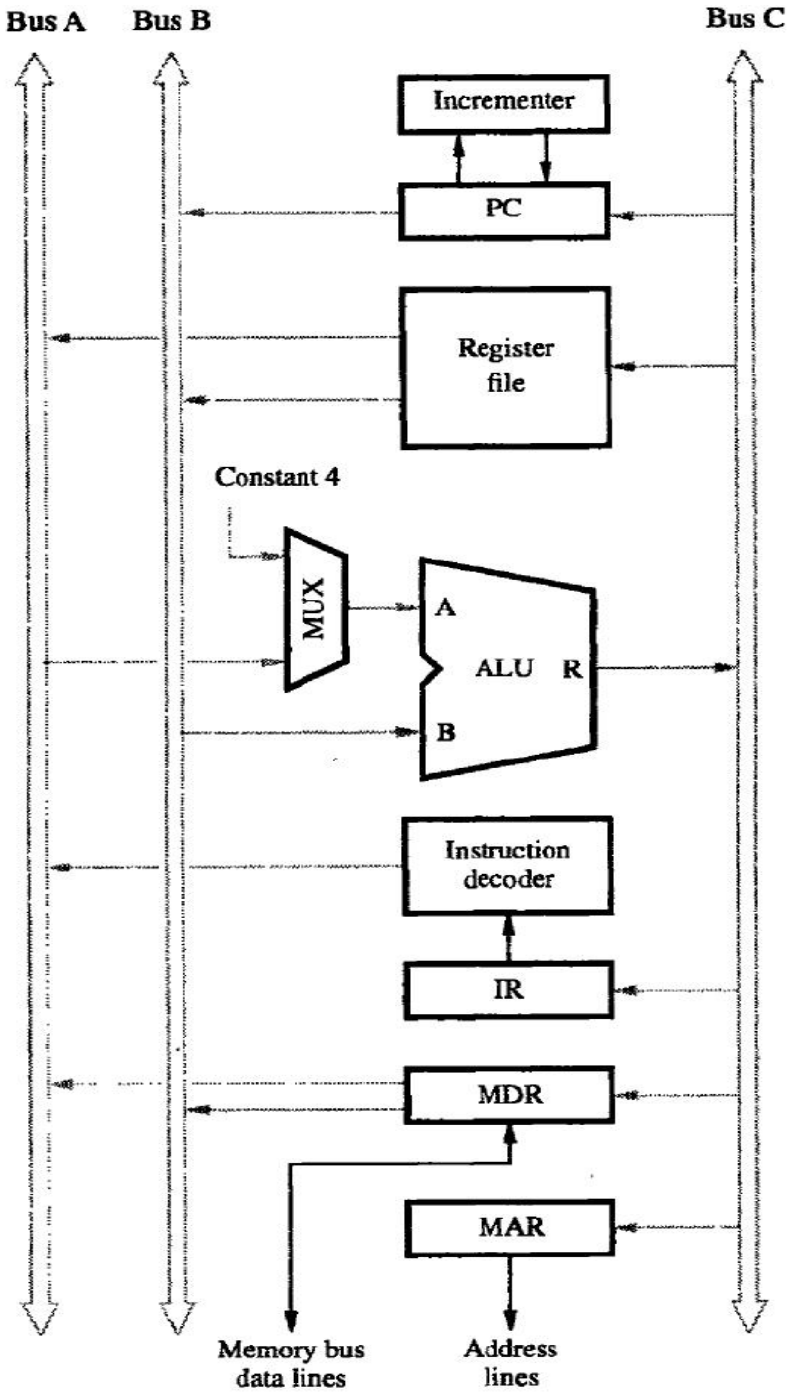
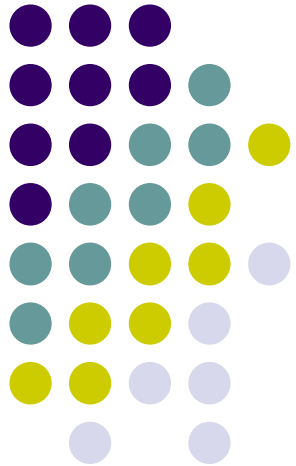
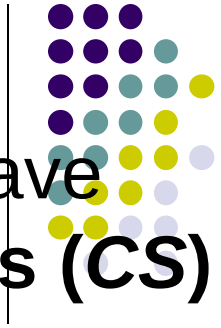


Figure 7.9. Control sequence for instruction: Add R4,R5,R6, for the three-bus organization of CPU Data Path (Figure 7.8)

Hardwired Control Unit

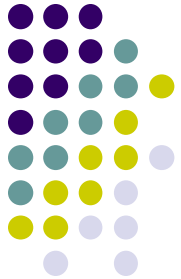


Overview



- To execute instructions, the processor must have some means of generating the **control signals (*CS*)** needed in the proper sequence.
- **Examples** of ***CS***: PC_{out}, PC_{in}, Ri_{out}, Rj_{in}, *ADD, SUB, MUL, DIV,* MAR_{in}, MDR_{out}, *SelectY,* Z_{in}, Z_{out}, *WMFC, End*, etc...
- Two paradigms/categories: hardwired control and micro programmed control
- Hardwired system can operate at high speed; but with little flexibility and more design complexity for large architecture/large no. of instructions.
- Micro programmed control unit provides enormous flexibility of design, allows simpler design, especially with larger architecture, but operates in much slower speed.

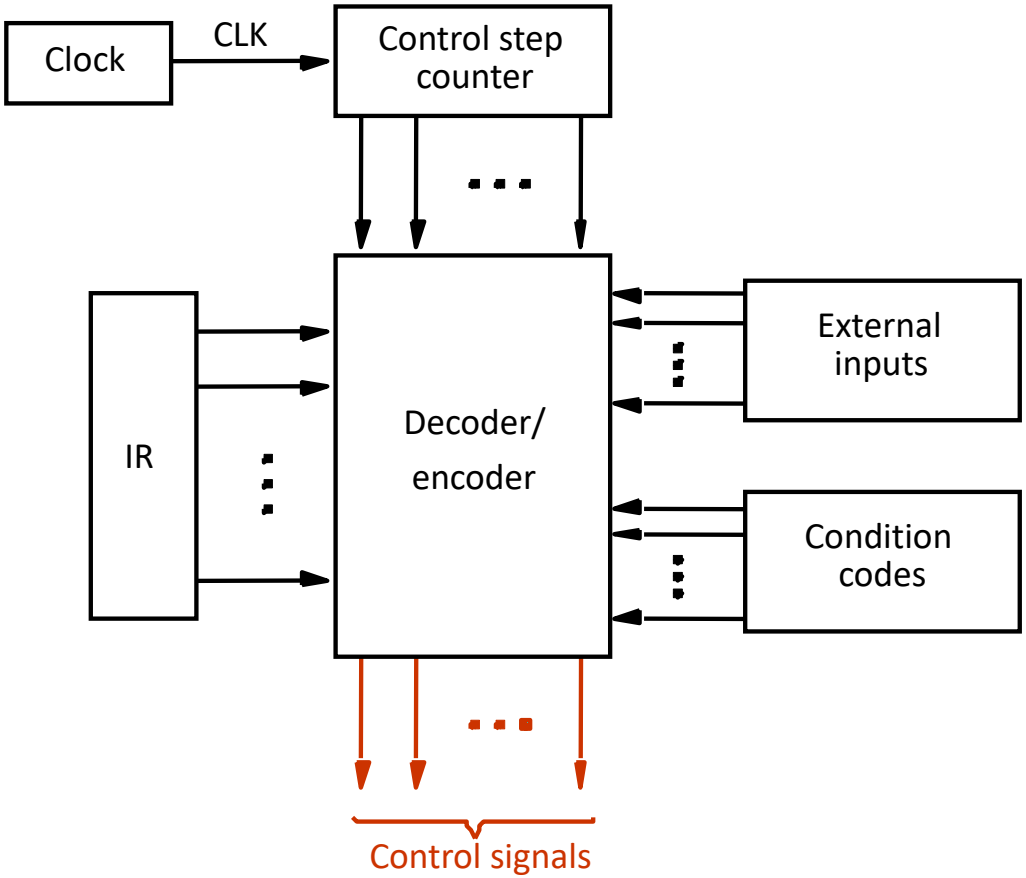
Hardwired control



- Required control signals are determined by the following information:
  - Contents of the control step counter.
    - Determines which step in the sequence.
  - Contents of the instruction register.
    - Determines the actual instruction
  - Contents of the condition code flags.
    - Used for example in a BRANCH instruction.
  - External input signals such as MFC.

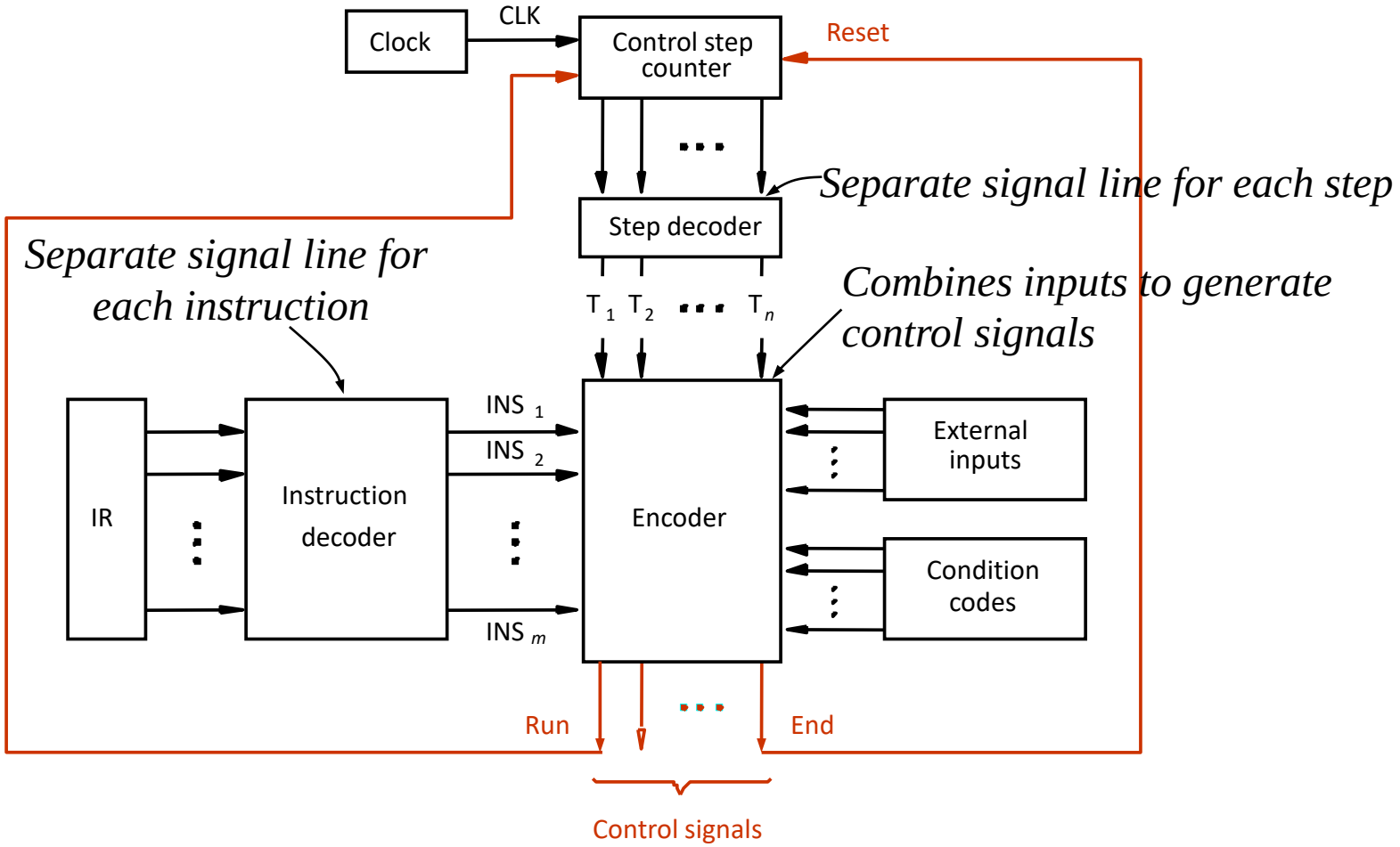
Hardwired control (contd..)

Control unit organization



- *Control unit consists of a decoder/encoder block to accept the following inputs:*
  - *Control step counter.*
  - *Instruction Register. IR*
  - *Condition codes*
  - *External inputs.*
- *Generates control signals.*

Hardwired control (contd..)

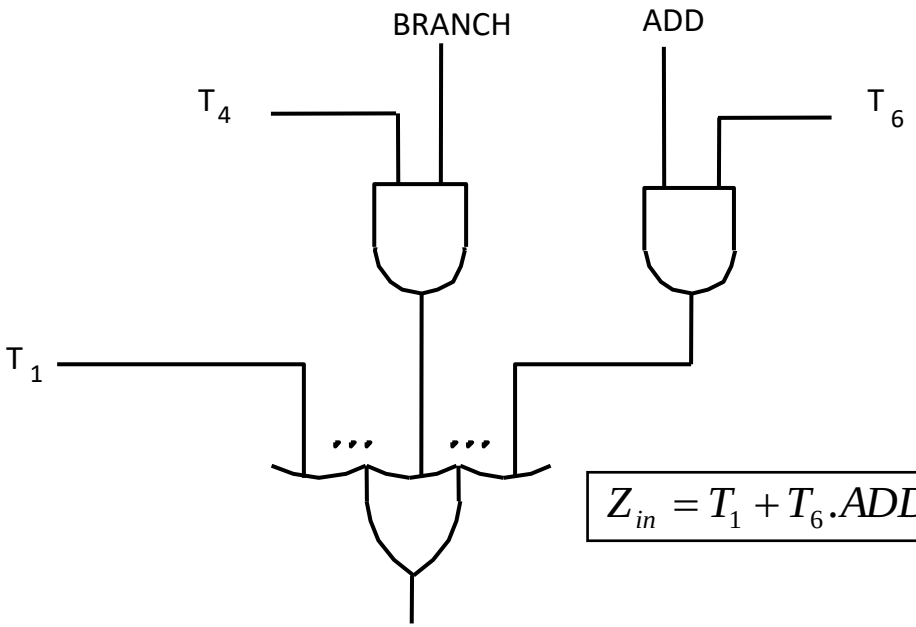


Hardwired control (contd..)

Control signals such as Z_{in}, PC_{out}, ADD are generated by encoder block

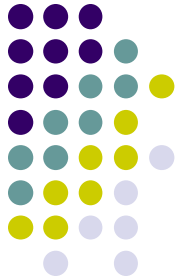
Suppose if Z_{in} is asserted:

- During T_1 for all instructions.
- During T_6 for ADD instruction.
- During T_4 for unconditional $BRANCH$ instruction
- .....



$$Z_{in} = T_1 + T_6.ADD + T_4.BRANCH + ...$$

Hardwired control (contd..)



- Control hardware can be viewed as a state machine:
  - Changes state every clock cycle depending on the contents of the instruction register, condition codes, and external inputs.
- Outputs of the state machine are control signals:
- Sequence of control signals generated by the machine is determined by wiring of logic elements, hence the name "hardwired control".
- Speed of operation is one of the advantages of hardwired control is its speed of operation.
- Disadvantages include:
  - Little flexibility.
  - Limited complexity of the instruction set it can implement.

A Complete Processor with Internal and external system bus (Self Study)

The System Has Two ALU's => The Integer and Floating point Units that operate on the Integer and floating point data values!!!

Separate Instruction and Data **cache** are for improved performance (hide the high Latency (delay) to access the memory).
Cache is discussed in details in the Memory chapter!!!

The Integer and Floating point Units are Full / Complete ALUs
with their own separate control units

Note the difference of the direction of arrows for Ins. Cache and Data Cache!

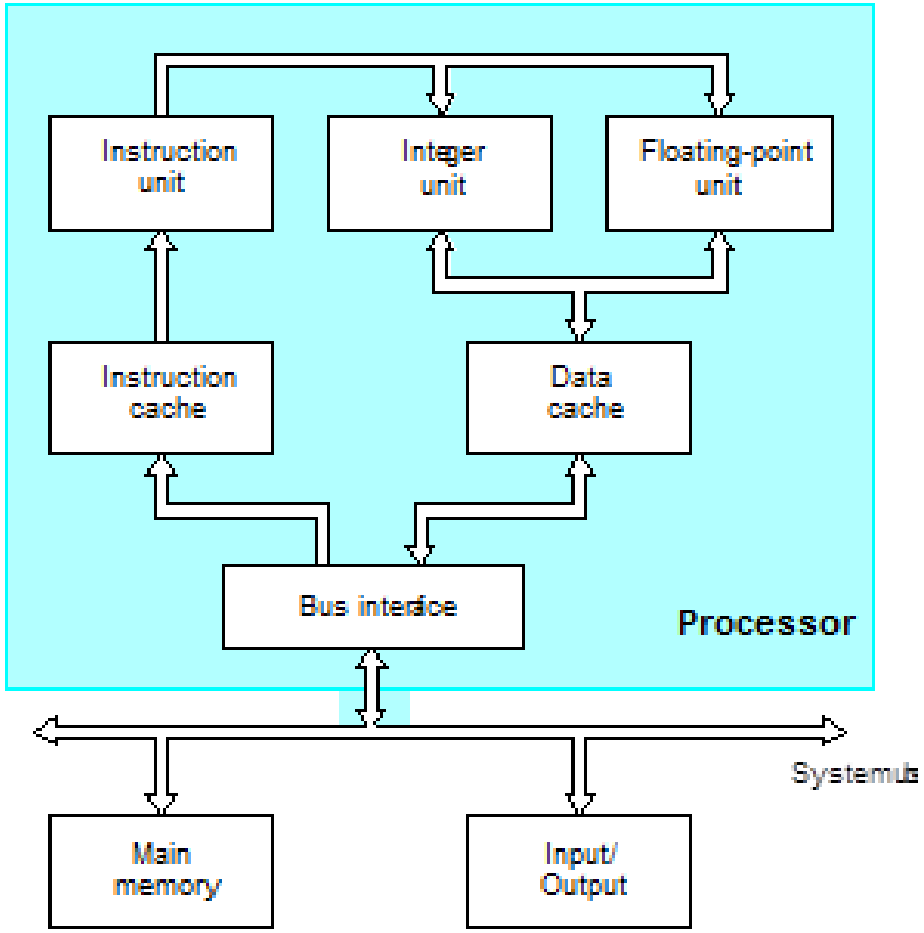
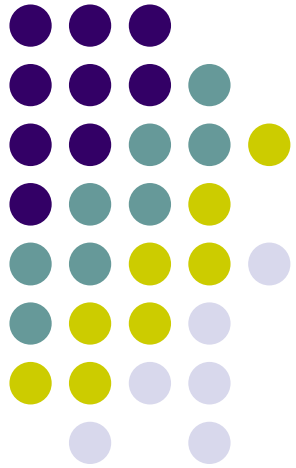
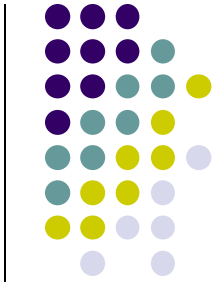


Figure 7.14. Block diagram of a complete processor

Microprogrammed Control Unit

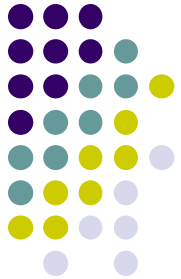


Microprogrammed control



- An alternative to Hardwired Control Unit
- Microprogrammed control:
  - Control signals are generated by a ***program*** similar to machine language programs.

Microprogrammed control (contd..)



Control Word (**CW**) is a word whose individual **bits** represent various control signals.

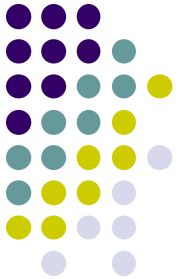
Step	Action
1	PC $_{out}$, MAR $_{in}$, Read, Select4, Add, Z $_{in}$
2	Z $_{out}$, PC $_{in}$, Y $_{in}$, WMF C
3	MDR $_{out}$, IR $_{in}$
4	R3 $_{out}$, MAR $_{in}$, Read
5	R1 $_{out}$, Y $_{in}$, WMF C
6	MDR $_{out}$, SelectY, Add, Z $_{in}$
7	Z $_{out}$, R1 $_{in}$, End

At every step, some control signals are asserted (=1) and all others are 0.

Control Signals:

- PC_{out}
- PC_{in}
- MAR_{in}
- ***Read***
- MDR_{out}
- IR_{in}
- Y_{in}
- ***SelectY***
- ***Select4***
- ***Add***
- Z_{in}
- Z_{out}
- $R1_{out}$
- $R1_{in}$
- $R3_{out}$
- ***WMFC***
- ***End .....***

Microprogrammed control (contd..)

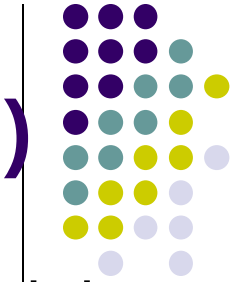


Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

- *At every **step**, a Control Word needs to be generated.*
- *Every **instruction** will need a sequence of CWs for its execution.*
- *Sequence of **CWs** for an **instruction** is the **microroutine** for the instruction.*
- *Each **CW** in this microroutine is referred to as a **microinstruction**.*

(SelectY is represented by Select=0, & Select4 by Select=1)

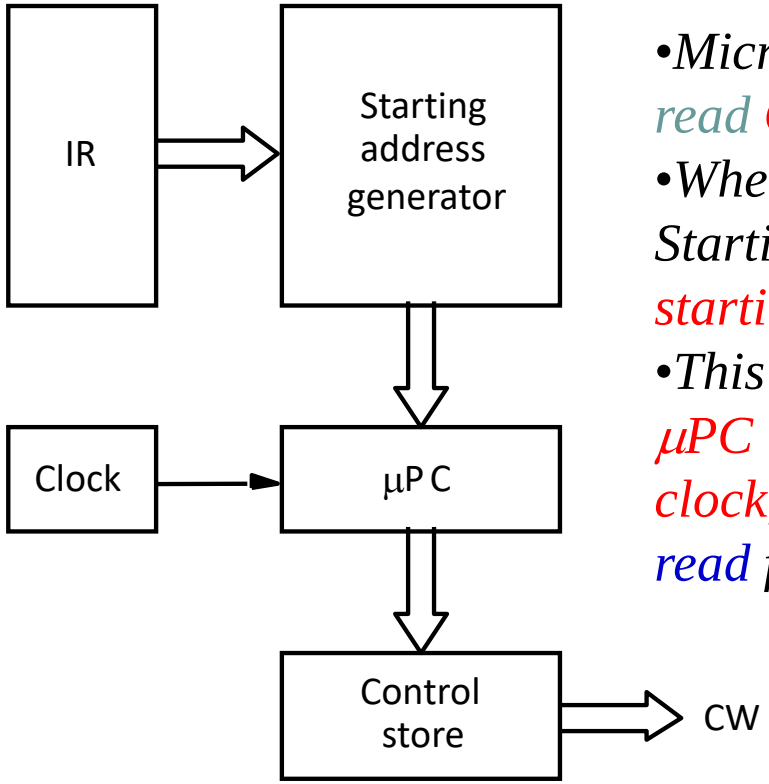
Microprogrammed control (contd..)



- Every instruction will have its own microroutine which is made up of microinstructions.
- Microroutines for **all** instructions in the instruction set of a computer are stored in a special memory called Control Store.
- Recall that the Control Unit generates the control signals:
  - Sequentially reading the CWs of the corresponding microroutine from the control store.

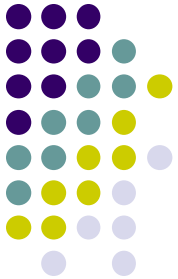
Microprogrammed control (contd..)

Basic organization of a microprogrammed control unit.



- *Microprogram counter (µPC) is used to read CWs from control store sequentially.*
- *When a new instruction is loaded into IR, Starting address generator generates the starting address of the microroutine.*
- *This address is loaded into the µPC. µPC is automatically incremented by the clock, so successive microinstructions are read from the control store.*

Self Study



Some basic functions cannot be carried out by this simple Organization, such as **Conditional Branching** and operations based on external inputs, such as **WMFC**

Self Study

If the CW is a conditional branch instruction, then check the conditional codes and if necessary, perform Branching by loading new address in µPC Using the offset-address-field of the IR register

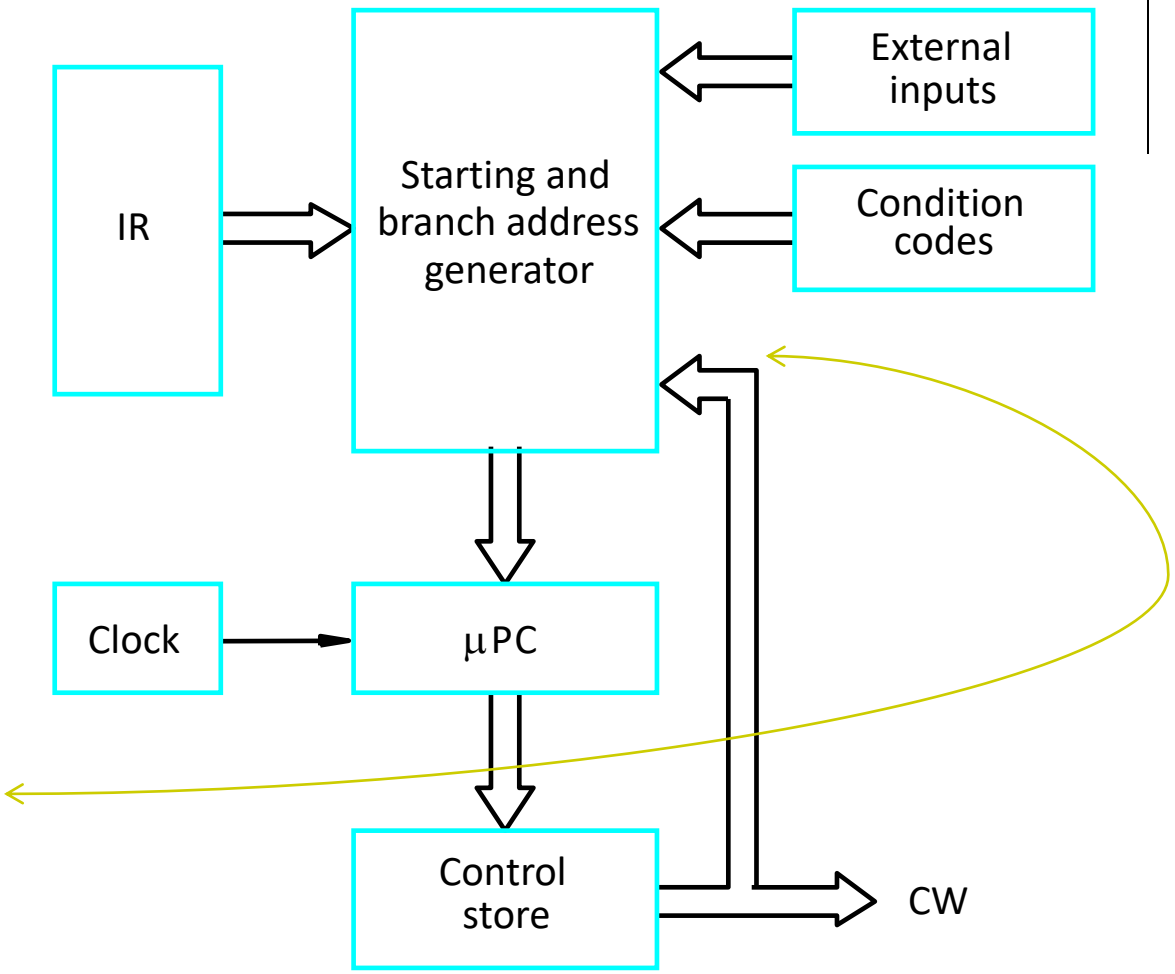


Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.