# Chapter 19 - Control Unit Operation

Luis Tarrataca
luis.tarrataca@gmail.com

CEFET-RJ

## Table of Contents I

## Table of Contents I

## Motivation

Remember when we talked about interruptions in Chapter 7?

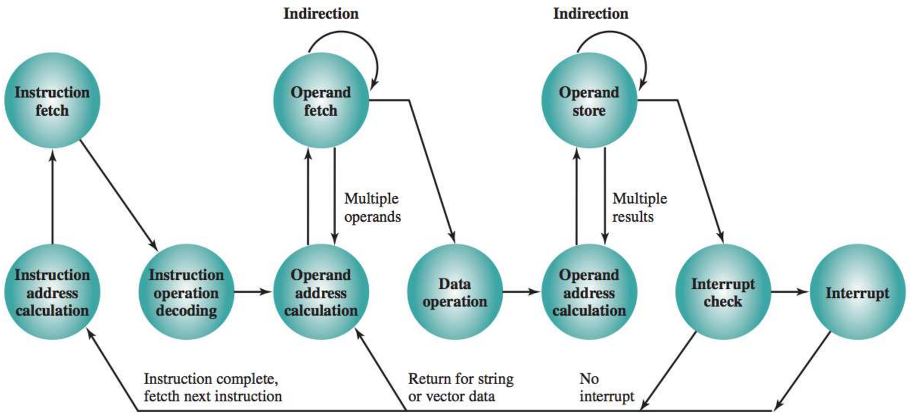

Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction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

Figure: Instruction Cycle State Diagram (Source: (Stallings, 2015))

Each instruction can be perceived as a sequence of operations:

- **Fetch Instruction**
- **Decode Instruction**
- **Fetch Operands**
- **Execute Instruction**
- **Operand Store**
- **Look for interrupts**

Today we will take a step further beyond these steps: **micro-operations**:

- Each one is very simple and accomplishes very little.

So, what is a micro-operation? Any ideas?

## Micro-operation:

- Program: sequence of instructions:
- Each instruction has multiple stages (fetch, decode, ..., interrupts):
  - Each one of these involves one or more $\mu$-operations;
  - These are atomic operations, not possible to go further than these;
  - Function decomposition FTW =)

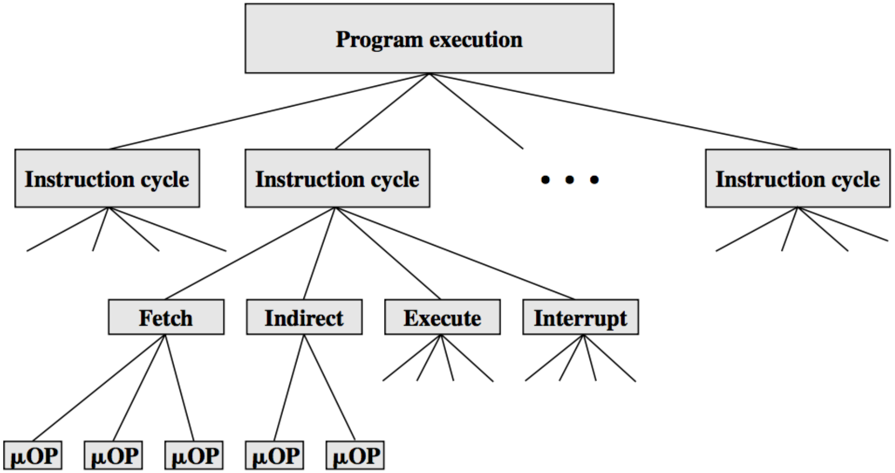
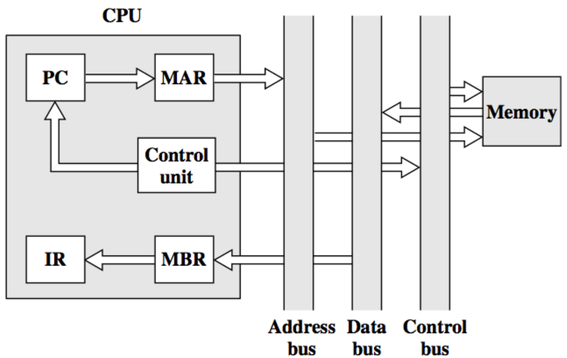


Figure: Constituent Elements of a Program Execution. (Source: (Stallings, 2015))

## $\mu$-operations

Assume the following organization:



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure: Data Flow Fetch Cycle (Source: (Stallings, 2015))

Four registers are involved (1/2):

- **Memory address register (MAR):**
    - Specifies the address in memory for a read or write operation
    - Connected to the address lines of the system bus
- **Memory buffer register (MBR):**
    - Contains one of two:
        - Value to be stored in memory;
        - Last value read from memory;
    - Connected to the data lines of the system bus,

Four registers are involved (2/2):

- **Program counter (PC):**
  - Contains the address of an instruction to be fetched.
- **Instruction register (IR):**
  - Contains the instruction most recently fetched.

Lets consider the **fetch** cycle:

What is the sequence of events regarding these processor registers? Any ideas?

## Fetch Cycle

Imagine that the PC has address 1100100

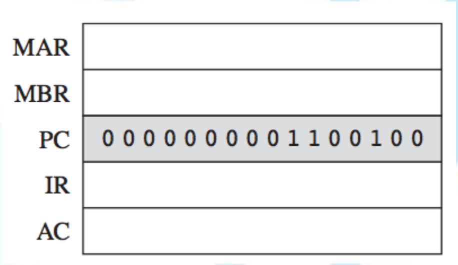


Figure: First step of the fetch - setup PC (Source: (Stallings, 2015))

- After the PC has been loaded what do we need to do?

## Fetch Cycle

Imagine that the PC has address 1100100

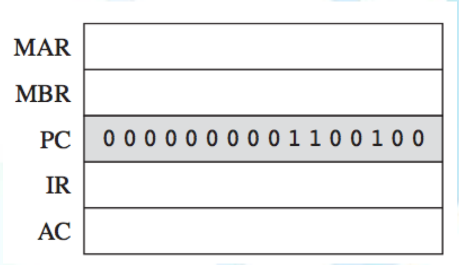


Figure: First step of the fetch - setup PC (Source: (Stallings, 2015))

- After the PC has been loaded what do we need to do?
  - PC holds the address of the next instructions;
  - We still need to load the instruction.

- Notice that the MAR is connected to the data lines of the bus;
- We need to copy the PC value to MAR and...;

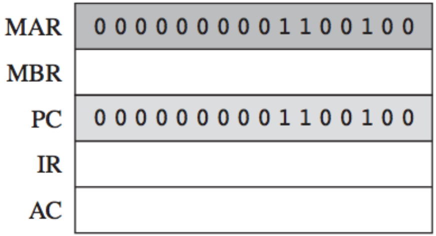


Figure: Second step of the fetch - setup MAR (Source: (Stallings, 2015))

- ...and issue a READ signal to the bus

- After the READ we will have the instruction on the MBR:

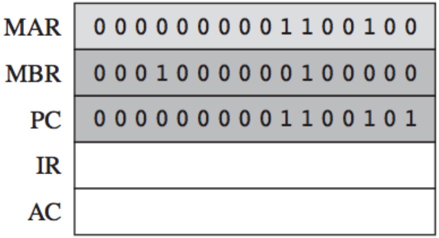
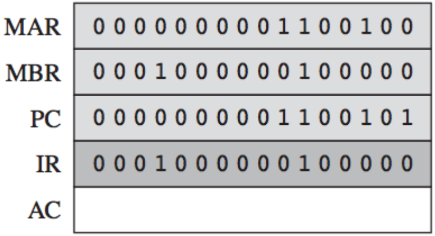


Figure: Third step of the fetch - setup MBR (Source: (Stallings, 2015))

- We also need to process the next instruction (PC+1);

- Cool, so what next?
  - But before we can do that we need to copy the instruction to the IR;



|  |  |
|---|---|
| MAR | 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| MBR | 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 |
| PC | 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 |
| IR | 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 |
| AC |  |

Figure: Fourth step of the fetch - setup IR (Source: (Stallings, 2015))

Thus the **fetch** cycle actually consists of:

- three steps and
- four micro-operations.

Symbolically, we can write this sequence of events as follows:

$$\begin{aligned} t_1 &: \text{MAR} \leftarrow \text{PC} \\ t_2 &: \text{MBR} \leftarrow \text{Memory} \\ & \quad \text{PC} \leftarrow \text{PC} + \text{I} \\ t_3 &: \text{IR} \leftarrow \text{MBR} \end{aligned}$$

Where $I$ is the instruction length.

This grouping would also have been valid:

$$
\begin{aligned}
t_1 &: \text{MAR} \leftarrow \text{PC} \\
t_2 &: \text{MBR} \leftarrow \text{Memory} \\
t_3 &: \text{PC} \leftarrow \text{PC} + \text{I} \\
&\quad \text{IR} \leftarrow \text{MBR}
\end{aligned}
$$

As long as:

- Proper sequence is followed;
- No read and write operations occur simultaneously on the same $t_k$;

Why is it possible to execute multiple instructions during time $t_2$ or $t_3$?

Each $\mu$-operation involves movement of data into or out of a register:

- If these movements do not interfere with one another:
  - several of them can take place during one step, saving time.

## Indirect cyle

Once an instruction is fetched, the next step is to **fetch source operands:**

- If the instruction specifies a direct address,

$$\begin{aligned} t_1 &: \text{MAR} \leftarrow \text{Address( IR )} \\ t_2 &: \text{MBR} \leftarrow \text{Memory} \end{aligned}$$

- If the instruction specifies an indirect address

$$\begin{aligned} t_1 &: \text{MAR} \leftarrow \text{IR( Address )} \\ t_2 &: \text{MBR} \leftarrow \text{Memory} \\ t_3 &: \text{Address( IR )} \leftarrow \text{MBR( Address )} \\ t_4 &: \text{MAR} \leftarrow \text{IR( Address )} \\ t_5 &: \text{MBR} \leftarrow \text{Memory} \end{aligned}$$

## Interrupt Cycle

Test to determine whether any interrupts have occurred:

- After the completion of the execute cycle.
- Example:

$$\begin{aligned}
t_1 &: \text{MBR} \leftarrow \text{PC} \\
t_2 &: \text{MAR} \leftarrow \text{SaveMemoryAddress} \\
&\phantom{:} \text{PC} \leftarrow \text{InterruptRoutineAddress} \\
&\vdots \\
t_n &: \text{MAR} \leftarrow \text{SaveMemoryAddress} \\
t_{n+1} &: \text{MBR} \leftarrow \text{Memory} \\
t_{n+2} &: \text{PC} \leftarrow \text{MBR}
\end{aligned}$$

- Logic:
  - Save the current PC into some memory location;
    - Where to? Stack Pointer usually contains an appropriate address;
  - Load the address of the interruption handling routine;
  - Execute the interruption handling routine;
  - Once the interruption handling routine finishes:
    - load the PC that was stored into memory.

## Execution Cycle

Fetch, indirect, and interrupt cycles are simple and predictable.

- Same micro-operations are repeated each time around.

This is not true of the execute cycle.

- Different opcodes possibilities imply different sequences.

ADD R1, X :

$t_1$ :MAR $\leftarrow$ IR( Address )
$t_2$ :MBR $\leftarrow$ Memory
$t_3$ :R1 $\leftarrow R1 + MBR$

ISZ X :

$t_1$ :MAR $\leftarrow$ IR( Address )
$t_2$ :MBR $\leftarrow$ Memory
MBR $\leftarrow$ MBR + 1
Memory $\leftarrow$ MBR
$t_3$ :If MBR == 0 then PC $\leftarrow$ PC + I

- ISZ = Increment and skip if zero;

## Instruction Cycle

We can now put all these stages into a single instruction cycle

- We will employ an auxiliary register: instruction cycle code (**ICC**).
- ICC designates the state of the processor:
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

At the end of each of the four cycles: **ICC** is set appropriately.

- Indirect cycle is always followed by the execute cycle.
- Interrupt cycle is always followed by the fetch cycle
- What about the fetch cycle?
  - Depends on whether direct or indirect addressing is used;
- What about the execute cycle?
  - Depends whether an interruption was activated;

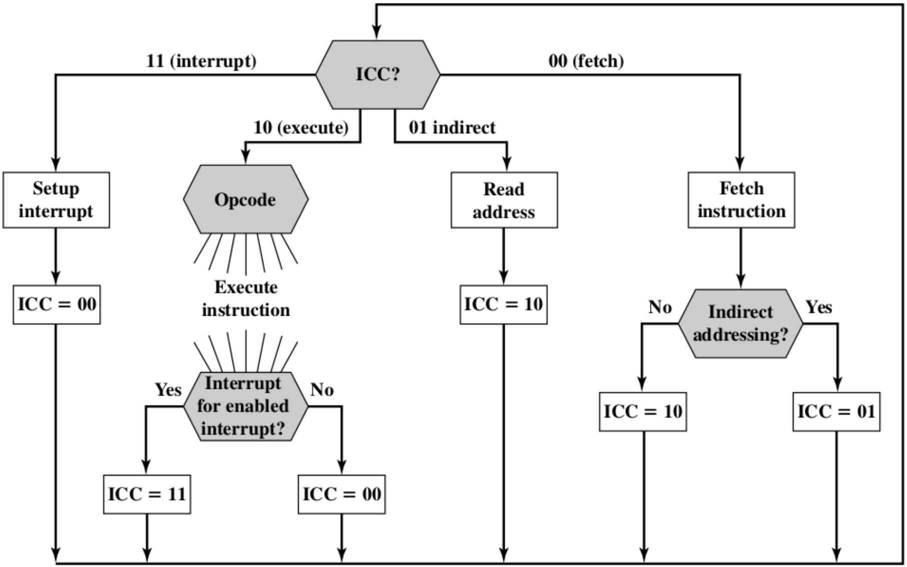


Figure: Flowchart for instruction cycle (Source: (Stallings, 2015))

## Control Unit Operation

$\mu$-operations fall into one of the following categories:

- Transfer data from one register to another;
- Transfer data from a register to an external interface (e.g., system bus);
- Transfer data from an external interface to a register;
- Perform an arithmetic operation;

What do you think will be the functional requirement of the control unit? Any ideas?

What do you think will be the functional requirement of the control unit? Any ideas?

Control unit performs two basic tasks:

- **Sequencing:** Go through a sequence of $\mu$-operations;
- **Execution:** causes each $\mu$-operation to be performed.

Lets have a look at the general diagram of a control unit:

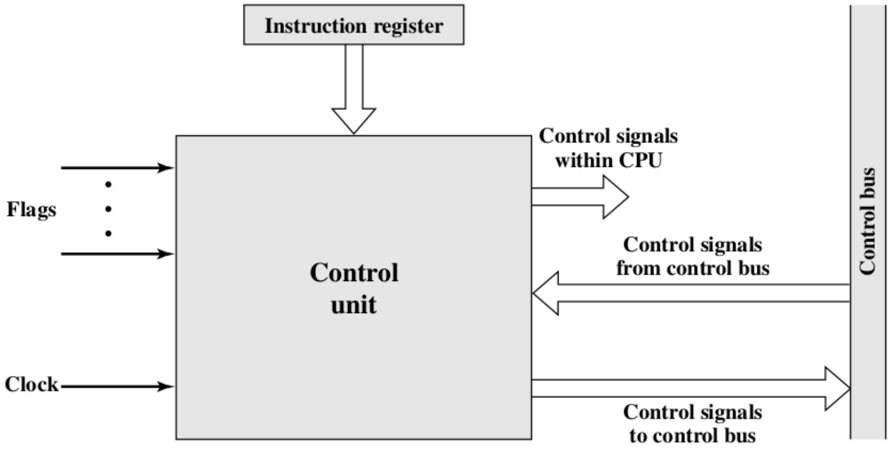

Instruction register
Control signals within CPU
Flags
Control unit
Control signals from control bus
Control bus
Clock
Control signals to control bus

Figure: Block diagram of the control unit (Source: (Stallings, 2015))

The **inputs** are:

- **Clock:**
  - Causes a set of simultaneous micro-operations to be performed;
- **Instruction register:**
  - Opcode and addressing mode to determine which $\mu$-operations to perform;
- **Flags:**
  - Indicating status of the processor and outcome of previous ALU operations;
- **Control signals:**
  - signals to the control unit.

The **outputs** are:

- Control signals **within** the processor:
  - Those that cause data to be moved from one register to another;
  - and those that activate specific ALU functions.
- Control signals to control bus:
  - Control signals to memory;
  - Control signals to the I/O modules.

## Example (1/3)

Consider **fetch** cycle, the control unit needs to:

- Transfer contents of the PC to MAR.
    - This is done by activating a control signal:
        - Opening the gates between the bits of PC and the bits of MAR.
- Read a word from memory into MBR and increment PC.

## Example (2/3)

Control unit sends the following control signals **simultaneously** (1/2):

- Control signal that opens gates:
  - Allowing contents of MAR onto the address bus;
- Memory read control signal on the control bus;
- Control signal that opens the gates:
  - Allowing contents of the data bus to be stored in MBR.

## Example (2/3)

Control unit sends the following control signals **simultaneously** (2/2):

- Control signals to logic thats
    - Adds 1 to the contents of PC;
    - Stores the result back to PC.

Then the control unit:

- Sends a control signal that opens gates between MBR and IR.

## Example (3/3)

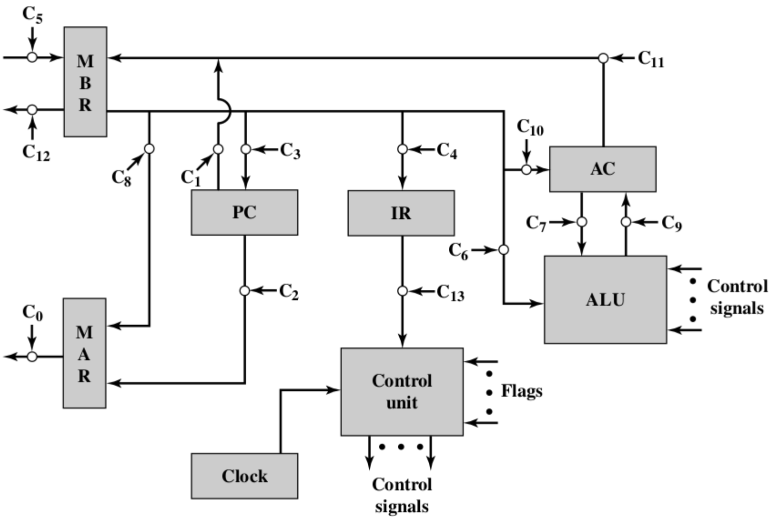


Figure: Data paths and control signals (Source: (Stallings, 2015))

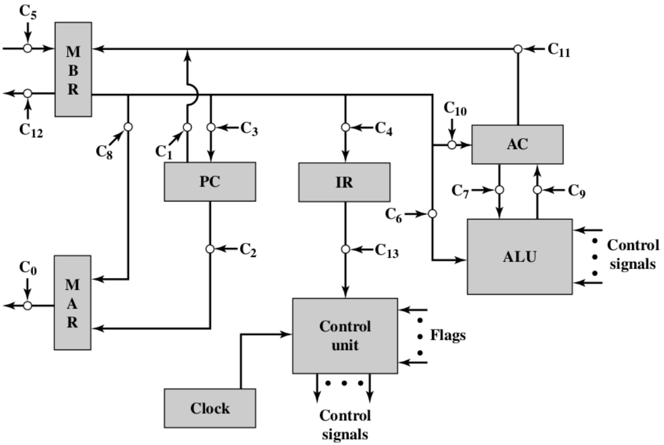
With each clock cycle control signals go to three destinations:

- **Data paths**, for each path to be controlled:
  - A switch $C_i$ temporarily opens the gate to let data pass;
- **ALU:**
  - Signals activate various logic circuits and gates within the ALU;
- **System bus:**
  - Control signals are sent out onto the control lines of the system bus;
  - *E.g.:* memory READ.

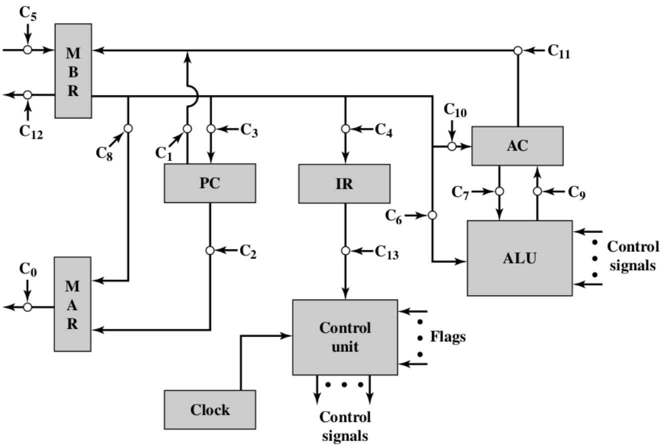
Control unit knows state of the instruction cycle:

- State-based sequence of control signals are emitted:
  - causing $\mu$-operations to occur;
- System clock pulses are used to time the sequence of events;

What are the control signals necessary for the fetch cycle? Any ideas?

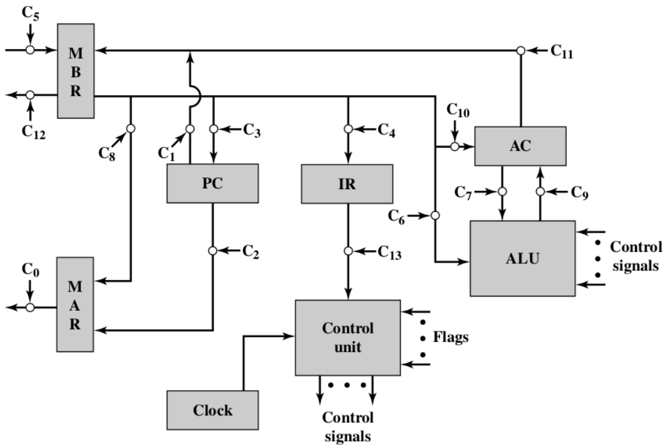


| Cycle | $\mu$-operations | Active Control Signals |
|---|---|---|
| Fetch | $t_1$ : MAR $\leftarrow$ (PC) | |
| | $t_2$ : MBR $\leftarrow$ Memory<br>PC $\leftarrow$ (PC) + 1 | |
| | $t_3$ : IR $\leftarrow$ (MBR) | |

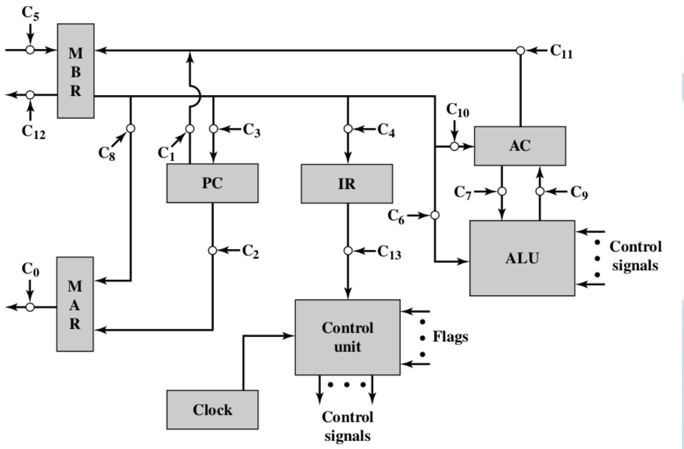


| Cycle | $\mu$-operations | Active Control Signals |
|---|---|---|
| Fetch | $t_1$ : MAR $\leftarrow$ (PC) | $C_2$ |
| | $t_2$ : MBR $\leftarrow$ Memory<br>PC $\leftarrow$ (PC) + 1 | $C_5, C_R$ |
| | $t_3$ : IR $\leftarrow$ (MBR) | $C_4$ |

What are the control signals necessary for the indirect cycle? Any ideas?

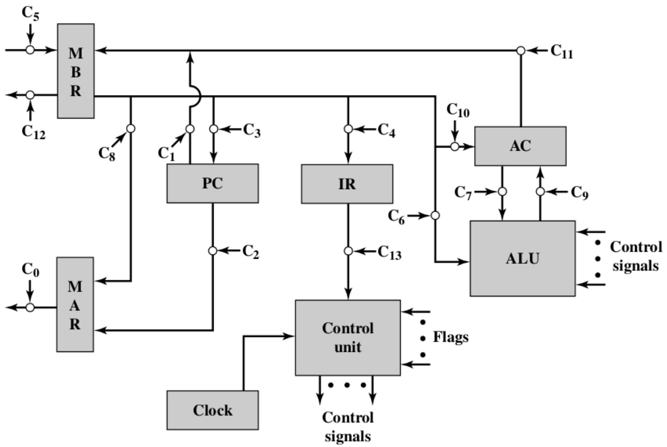


| Cycle | $\mu$-operations | Active Control Signals |
|---|---|---|
| Indirect | $t_1$ : MAR ← (IR(Address)) | |
| | $t_2$ : MBR ← Memory | |
| | $t_3$ : IR(Address) ← (MBR(Address)) | |

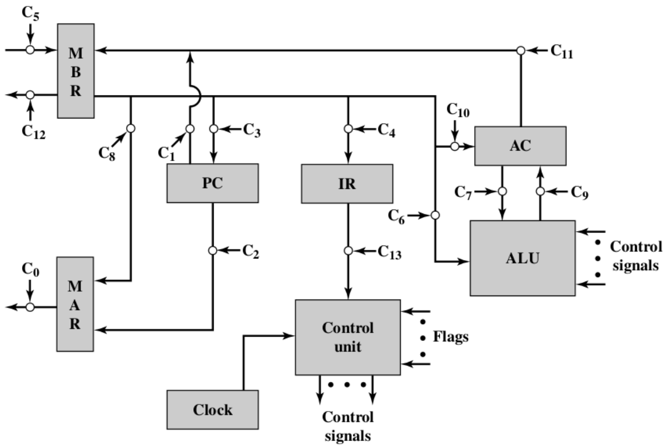


| Cycle | $\mu$-operations | Active Control Signals |
|---|---|---|
| Indirect | $t_1$ : MAR ← (IR(Address)) | $C_8$ |
| | $t_2$ : MBR ← Memory | $C_5$, $C_R$ |
| | $t_3$ : IR(Address) ← (MBR(Address)) | $C_4$ |

What are the control signals necessary for the interrupt cycle? Any ideas?

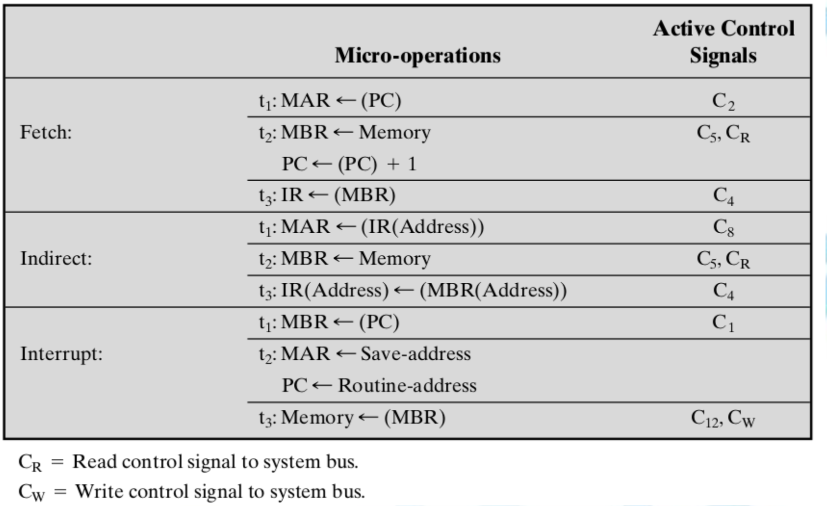


| Cycle | $\mu$-operations | Active Control Signals |
|---|---|---|
| Interrupt | $t_1$ : MBR $\leftarrow$ (PC) | |
| | $t_2$ : MAR $\leftarrow$ Save-address<br>PC $\leftarrow$ Routine-address | |
| | $t_3$ : Memory $\leftarrow$ (MBR) | |



| Cycle | $\mu$-operations | Active Control Signals |
|---|---|---|
| Interrupt | $t_1$ : MBR $\leftarrow$ (PC) | $C_1$ |
| | $t_2$ : MAR $\leftarrow$ Save-address<br>PC $\leftarrow$ Routine-address | |
| | $t_3$ : Memory $\leftarrow$ (MBR) | $C_{12}$, $C_W$ |

## Conclusions



| | Micro-operations | Active Control Signals |
|---|---|---|
| Fetch: | $t_1$: MAR ← (PC) | $C_2$ |
| | $t_2$: MBR ← Memory<br>PC ← (PC) + 1 | $C_5$, $C_R$ |
| | $t_3$: IR ← (MBR) | $C_4$ |
| Indirect: | $t_1$: MAR ← (IR(Address)) | $C_8$ |
| | $t_2$: MBR ← Memory | $C_5$, $C_R$ |
| | $t_3$: IR(Address) ← (MBR(Address)) | $C_4$ |
| Interrupt: | $t_1$: MBR ← (PC) | $C_1$ |
| | $t_2$: MAR ← Save-address<br>PC ← Routine-address | |
| | $t_3$: Memory ← (MBR) | $C_{12}$, $C_W$ |

$C_R$ = Read control signal to system bus.
$C_W$ = Write control signal to system bus.

Figure: $\mu$-operations and control signals (Source: (Stallings, 2015))

## Hardwired implementation

Now that we know what functions the control unit must perform:

How can we implement the control unit?

Wide variety of techniques have been used, usually one of:

- Hardwired implementation;
- Microprogrammed implementation.

Lets take a brief look at the hardwired implementation:

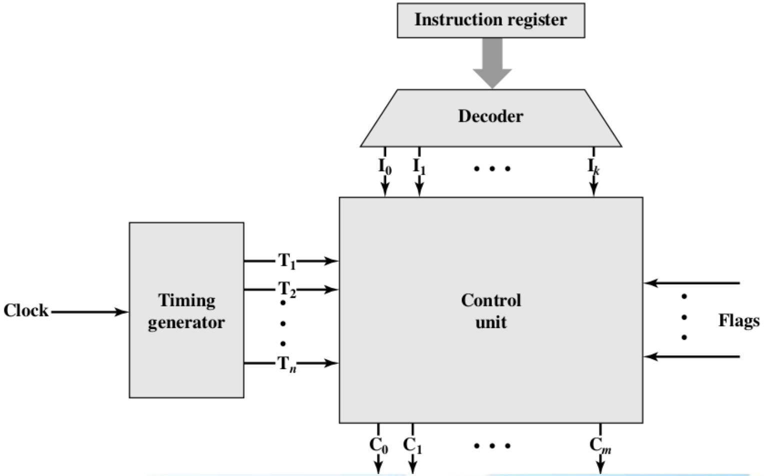


Figure: Control unit with decoded inputs (Source: (Stallings, 2015))

## Example

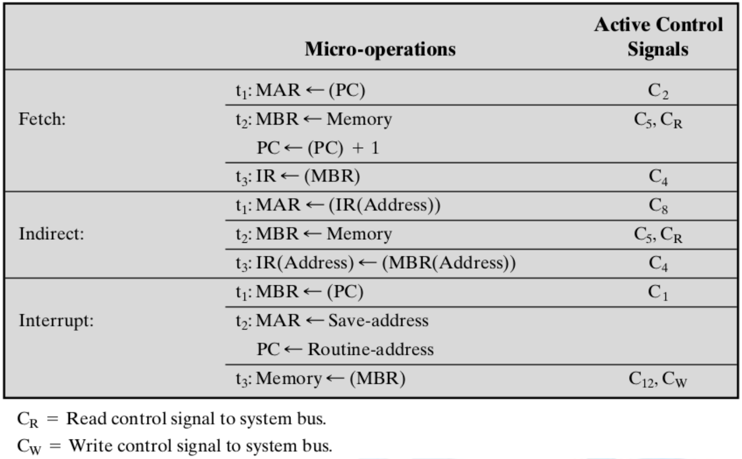
Lets use the Table from Slide 27:

- With a hardwired implementation:
  - Boolean expressions need to be derived as a function of the inputs.
- Define two new control signals P and Q:

| PQ | Description |
|---|---|
| 00 | Fetch cycle |
| 01 | Indirect cycle |
| 10 | Execute cycle |
| 11 | Interrupt cycle |

Then the following Boolean expression defines $C_5$:

$$C_5 = \overline{P}\,\overline{Q}T_2 + \overline{P}QT_2$$



| | Micro-operations | Active Control Signals |
|---|---|---|
| Fetch: | $t_1$: MAR ← (PC) | $C_2$ |
| | $t_2$: MBR ← Memory<br>PC ← (PC) + 1 | $C_5$, $C_R$ |
| | $t_3$: IR ← (MBR) | $C_4$ |
| Indirect: | $t_1$: MAR ← (IR(Address)) | $C_8$ |
| | $t_2$: MBR ← Memory | $C_5$, $C_R$ |
| | $t_3$: IR(Address) ← (MBR(Address)) | $C_4$ |
| Interrupt: | $t_1$: MBR ← (PC) | $C_1$ |
| | $t_2$: MAR ← Save-address<br>PC ← Routine-address | |
| | $t_3$: Memory ← (MBR) | $C_{12}$, $C_W$ |

$C_R$ = Read control signal to system bus.
$C_W$ = Write control signal to system bus.

Figure: Micro-operations and control signals (Source: (Stallings, 2015))

Then the following Boolean expression defines $C_5$:

$$C_5 = \overline{PQ}T_2 + \overline{P}QT_2$$

Is this enough for defining $C_5$?

Then the following Boolean expression defines $C_5$:

$$C_5 = \overline{PQ}T_2 + \overline{P}QT_2$$

Is this enough for defining $C_5$?

- Not enough: $C_5$ is also needed during the execute cycle.

Assume that there are only three instructions that read from memory:

- **LDA**;
- **ADD**;
- **AND**.

Now we can define $C_5$ as:

$$C_5 = \overline{PQ}T_2 + \overline{P}QT_2 + P\overline{Q}(LDA + ADD + AND)T_2$$

Process would have to be repeated for every control signal:

- Resulting in a set of Boolean equations defining control unit behaviour.

Can you see any problems with the hardwired version?

Process would have to be repeated for every control signal:

- Resulting in a set of Boolean equations defining control unit behaviour.

Can you see any problems with the hardwired version?

In a modern processor:

- Number of boolean equations defining control unit behaviour is very large:
- Implementing a circuit satisfying these equations is extremely complex.

So what is the alternative to hardwired implementations? Any ideas?

So what is the alternative to hardwired implementations? Any ideas?

- Hint 1: We have used the same concepts before...

So what is the alternative to hardwired implementations? Any ideas?

- Hint 1: We have used the same concepts before...
- Hint 2: Concepts where used when we studied boolean circuits...

So what is the alternative to hardwired implementations? Any ideas?

- Hint 1: We have used the same concepts before...
- Hint 2: Concepts where used when we studied boolean circuits...
- Hint 3: Von...

So what is the alternative to hardwired implementations? Any ideas?

- Hint 1: We have used the same concepts before...
- Hint 2: Concepts where used when we studied boolean circuits...
- Hint 3: Von Neumann architecture, *i.e.:* programmable computer;

Idea: Apply Von Neumann architecture concepts to $\mu$-operations:

- known as $\mu$-programming;
- This is the subject of the next chapter =)

## Where to focus your study

After this class you should be able to understand that:

- The execution of an instruction involves the execution of a sequence of substeps, generally called cycles;
- Each cycle is in turn made up of a sequence of more fundamental operations, called micro-operations;
- The control unit causes the processor to step through a series of micro-operations in the proper sequence and generates the appropriate control signals;

Less important to know how these solutions were implemented:

- details of specific hardware solutions.

Your focus should always be on the building blocks for developing a solution =)

## References I

Stallings, W. (2015).

*Computer Organization and Architecture: Designing for Performance*.

Pearson Education, 10th edition edition.