Micro-Operations

• Instruction execution
  ➢ execution of a sequence of steps, i.e., cycles
• Fetch, Indirect, Execute & Interrupt cycles
• Cycle - a sequence of micro-operations
• Micro-operations
  ➢ data transfer between registers
  ➢ transfer between a register & an external bus
  ➢ ALU operation
Control Unit Atomic Operations

- CU causes the processor to step through a series of micro-operations in the proper sequence
- CU generates the control signals that cause each micro-operation to be executed
- Micro-Operations are the **atomic operations** of the Processor
- Control signals open & closes the logic gates
  - transfer of data to & from the registers
  - operation of the ALU
- Implementation of Control Unit
  - Hardwired
  - Microprogrammed, i.e., microinstructions
Constituent Elements of Program Execution

Diagram showing the constituent elements of program execution, including Instruction Cycle, Fetch, Indirect, Execute, and Interrupt, with their respective µOP operations.
Functional Specifications of a Processor

- Operations
- Addressing Modes Defined by the Instruction Set
- Registers – user visible
- I/O Module Interface Defined by the System
- Memory Module Interface Bus Specification
- Interrupts Defined by System Bus Specification & Operating System Support

Execution Time Sequence of Instructions is not necessarily the same as the Written Sequence of Instructions in the Program (branching instructions)
Fetch - 4 Registers

• Memory Address Register (MAR)
  – Connected to address lines of system bus
  – Specifies address for read or write operation

• Memory Buffer Register (MBR)
  – Connected to data lines of system bus
  – Holds data to write or last data read

• Program Counter (PC)
  – Holds address of next instruction to be fetched

• Instruction Register (IR)
  – Holds last instruction fetched
Fetch Sequence

- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory) \([\text{micro-code RISC, length} == 1]\)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches
Fetch Sequence (symbolic)

• t1: MAR <- (PC); CU issues \texttt{READ} command
• t2: MBR <- (memory) simultaneously
  PC <- (PC) +1
• t3: IR <- (MBR)
  where \( t_x \) refers to the time unit/clock cycle

------------------ or ------------------

• t1: MAR <- (PC)
• t2: MBR <- (memory)
• t3: PC <- (PC) +1
  IR <- (MBR)
Interrupt Cycle — completion of the execute cycle
  — test for interrupt occurrences -- process pending interrupts

• t1: MBR ← (PC)

• t2: MAR ← Save_Address for PC contents
  PC ← address of start of interrupt processing routine,
  i.e., the interrupt Processing Routine_Address
  ♦ Multiple types and/or levels of interrupts, hence
  ♦ additional micro-operations may be required to obtain both
  ♦ Save_Address & Routine_Address

• t3: memory ← (MBR) - actual saving of the PC contents

• Saving the context is done by interrupt handler routine, not
  micro-ops
Execute Cycle (ADD)

• Different sequence of micro-operations for each instruction

• **ADD R1, X** - add the contents of location X to Register 1, place the result in R1

  • t1: MAR ← (IR(address(X)))
  • t2: MBR ← (Memory)
  • t3: R1 ← (R1) + (MBR)

  • Note: there is no overlap of micro-operations
Execute Cycle (ISZ)

- ISZ X - increment and skip if zero
  - t1:  \( \text{MAR} \leftarrow (\text{IR}(\text{address}(x))) \)
  - t2:  \( \text{MBR} \leftarrow (\text{memory}) \)
  - t3:  \( \text{MBR} \leftarrow (\text{MBR}) + 1 \)
  - t4:  \( \text{memory} \leftarrow (\text{MBR}) \)

if (MBR) == 0 then PC \( \leftarrow (\text{PC}) + 1 \)

Test & action operation is one micro op performed during time unit t4
**Execute Cycle (BSA)**

**BSA X - Branch and save address**

- Subroutine call instruction
- Address of instruction following BSA is saved in X; it will be used to return from the subroutine
- Execution continues from X+1

- **t1:** $\text{MAR} \leftarrow \text{IR(address(X))}$
- **t2:** $\text{PC} \leftarrow \text{IR(address(X))}$
- **t3:** $\text{PC} \leftarrow \text{PC} + 1$

$X$: return address

$X+1$: start of subroutine

... 

... 

$X+n$: return from subroutine
Instruction Cycle

• Each phase decomposed into sequence of elementary micro-operations
• E.g. fetch, indirect, and interrupt cycles
• Execute cycle
  – One sequence of micro-operations for each opcode
• Need to tie sequences together
• Assume new 2-bit register
  – Instruction cycle code (ICC) designates which part of cycle processor is in
    • 00: Fetch
    • 01: Indirect
    • 10: Execute
    • 11: Interrupt
Flowchart for Instruction Cycle (Code)

Indirect Cycle ➔ Execute Cycle ➔ next cycle depends upon the state of the system
Interrupt Cycle ➔ Fetch Cycle ➔ next cycle depends upon the state of the system
Types of Micro-operation

• Transfer data between registers

• Transfer data from register to external interface

• Transfer data from external interface to register

• Perform arithmetic or logical operations using registers for I/O
Functions of Control Unit using Control Signals

• Sequencing
  – CU causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed

• Execution
  – CU causes each micro-operation to be performed

• Control Signals
  – External: inputs indicating the state of the system
  – Internal: logic required to perform the sequencing and execution functions
Control Signals

• **Clock** (clock cycle time, processor cycle time)
  – One micro-instruction (or set of parallel micro-instructions) per clock cycle

• **Instruction register**
  – Opcode & addressing mode for current instruction
  – Determines which micro-instructions are performed

• **Flags** — Used to determine the
  – status of the CPU
  – Results of previous ALU operations

• **Signals from the control bus part of the system bus**
  – Interrupts
  – Acknowledgements
Model of Control Unit

Instruction register

Control Unit

Flags

Clock

Control signals within CPU

Control signals from control bus

Control signals to control bus
Control Signals - output

• Control Signals within CPU
  – Cause data movement register to register
  – Activate specific ALU functions
  – Activate a specific data path

• Control Signals to the control bus
  – To memory via the system bus
  – To I/O modules
Control Signal Sources

- **Clock**
  - One micro-instruction (or set of parallel micro-instructions) per clock cycle

- **Instruction Register**
  - Op-code for current instruction
  - Determines which micro-instructions are performed

- **Flags**
  - State of CPU
  - Results of previous operations

- **From Control Bus**
  - Interrupts / Bus Requests
  - Acknowledgements
Control Signals Outputs

• Within CPU
  – Cause data movement
  – Activate specific functions

• Via Main Bus
  – To memory
  – To I/O modules
Control Signals

### Micro-operations

<table>
<thead>
<tr>
<th>Micro-operations</th>
<th>Timing</th>
<th>Active Control Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_1$: MAR ← (PC)</td>
<td>$C_2$</td>
<td></td>
</tr>
<tr>
<td>$t_2$: MBR ← Memory</td>
<td>$C_3$, $C_R$</td>
<td></td>
</tr>
<tr>
<td>$t_3$: PC ← (PC) + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indirect:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_1$: MAR ← (IR(Address))</td>
<td>$C_9$</td>
<td></td>
</tr>
<tr>
<td>$t_2$: MBR ← Memory</td>
<td>$C_5$, $C_R$</td>
<td></td>
</tr>
<tr>
<td>$t_3$: IR(Address) ← (MBR(Address))</td>
<td>$C_4$</td>
<td></td>
</tr>
<tr>
<td>Interrupt:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_1$: MBR ← (PC)</td>
<td>$C_1$</td>
<td></td>
</tr>
<tr>
<td>$t_2$: MAR ← Save-address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_3$: Memory ← (MBR)</td>
<td>$C_{12}$, $C_W$</td>
<td></td>
</tr>
</tbody>
</table>

$C_R$ = Read control signal to system bus.
$C_W$ = Write control signal to system bus.
The Internal Bus ?
Example Simple Processor & Data Paths
Example Simple Processor & Data Paths
State Machine for Example Simple Processor

Control Unit with Decoded Inputs
Problems With Hard Wired Designs

• Sequencing & micro-operation logic gets complex

• Difficult to design, prototype, and test

• Resultant design is inflexible, and difficult to build upon (Pipeline, multiple computation units, etc.)

• Adding new instructions requires major design and adds complexity quickly
Example Simple Processor Micro-Programmed Control
Control Unit Organization

The Control Memory contains sequences of microinstructions that provide the control signals to execute instruction cycles, e.g. Fetch, Indirect, Execute, and Interrupt.

Tasks of Control Unit:
- Microinstruction sequencing
- Microinstruction execution

May be expected to complete instruction execution in “1” clock cycle. How is this possible?
Recall: Micro-sequencing

<table>
<thead>
<tr>
<th>Micro-operations</th>
<th>Timing</th>
<th>Active Control Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fetch:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_1$: MAR ← (PC)</td>
<td>$C_2$</td>
</tr>
<tr>
<td></td>
<td>$t_2$: MBR ← Memory</td>
<td>$C_5, C_R$</td>
</tr>
<tr>
<td></td>
<td>PC ← (PC) + 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_3$: IR ← (MBR)</td>
<td>$C_4$</td>
</tr>
<tr>
<td><strong>Indirect:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_1$: MAR ← (IR(Address))</td>
<td>$C_8$</td>
</tr>
<tr>
<td></td>
<td>$t_2$: MBR ← Memory</td>
<td>$C_5, C_R$</td>
</tr>
<tr>
<td></td>
<td>$t_3$: IR(Address) ← (MBR(Address))</td>
<td>$C_4$</td>
</tr>
<tr>
<td><strong>Interrupt:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_1$: MBR ← (PC)</td>
<td>$C_1$</td>
</tr>
<tr>
<td></td>
<td>$t_2$: MAR ← Save-address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PC ← Routine-address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_3$: Memory ← (MBR)</td>
<td>$C_{12}, C_W$</td>
</tr>
</tbody>
</table>

$C_R$ = Read control signal to system bus.
$C_W$ = Write control signal to system bus.
Example of Control Memory Organization

Microinstructions:
- Generate Control Signals
- Provide Branching

- Fetch cycle routine
- Indirect cycle routine
- Interrupt cycle routine
- Execute cycle beginning
- AND routine
- ADD routine
- IOF routine
Horizontal vs Vertical Microprogramming

Horizontal Microprogrammed or
- Unpacked
- Hard
- Direct

Vertical Microprogrammed or
- Packed
- Soft
- Indirect
Microinstruction Encoding - Direct Encoding

(a) Direct encoding

Control signals

Decode logic

Field

Field

Field

...
Microinstruction Encoding - Indirect Encoding

(b) Indirect encoding
Horizontal Micro-programming

• Wide control memory word
• High degree of parallel operations possible
• Little encoding of control information
• Faster
Vertical Micro-programming

• Width can be much narrower

• Control signals encoded into function codes – need to be decoded

• More complex, more complicated to program, less flexibility

• More difficult to modify

• Slower
Typical Microinstruction Formats

(a) Horizontal microinstruction

(b) Vertical microinstruction
Example Microprogramming Formats

- MicroProgram Counter
- Subroutines
- Stack
- Control Register (MicroProgram Format)
Next Address Decision

- Depending on ALU flags and control buffer register:
  - Get next instruction
    - Add 1 to control address register
  - Jump to new routine based on jump microinstruction
    - Load address field of control buffer register into control address register
  - Jump to machine instruction routine
    - Load control address register based on opcode in IR
Microprogrammed Control Unit
Design Considerations

• Necessity of *speed*

• Size of Microinstructions

• Address generation
  – Branches
    • Both conditional and unconditional
    • Based on current microinstruction, condition flags, contents of IR
  • Based on format of address information
    – Two address fields
    – Single address field
Branch Control: Two Address Fields

Branch based upon:

- Instruction Opcode
- Address 1
- Address 2

Does require a wide microinstruction, but no address calculation is needed
Branch Control: Single Address Field

Branch based upon:
- Next instruction
- Address
- Opcode

Does require more circuitry, e.g. adder
Branch Control: Variable Format

One bit determines microinstruction format:

- Control signal format
- Branch format

Does require even more circuitry, and is slowest.
Advantages and Disadvantages of Microprogramming

Advantage:
• Simplifies design of control unit
  – Cheaper to design
  – Less error-prone
  – Much easier to modify
  – Supports having multiple versions / models

Disadvantage:
• Slower
• More expensive to produce in quantities