Chapter 16
Control Unit Operations
Execution of the Instruction Cycle

- It has many elementary phases, each executed in a single clock cycle (remember pipelining).
- In each phase only very simple operations (called *micro-operations*) are executed:
  - Move contents between registers (internals, interface with ALU, interface with memory)
  - Activate devices (ALU, memory)
- Micro-operations are the CPU atomic operations, hence define its low-level behaviour.
- A micro-operation is the set of actions (data flows and controls) that can be completed in a single clock cycle.
Constituent Elements of Program Execution

Diagram:

- Program Execution
  - Instruction Cycle
    - Fetch
      - μOP
      - μOP
    - Indirect
    - Execute
    - Interrupt
      - μOP
      - μOP
      - μOP
Sequence of micro-operations for instruction fetch

- \( t_1 \): \( \text{MAR} \leftarrow \text{PC} \) \(<\text{DF}_1>\)
- \( t_2 \): \( \text{MBR} \leftarrow \text{memory} \) \(<\text{DF}_2 \text{ DF}_3 \text{ DF}_4 \text{ DF}_5>\)
  \( \text{PC} \leftarrow \text{PC} + 1 \) \(<\text{DF}_7>\)
- \( t_3 \): \( \text{IR} \leftarrow \text{MBR} \) \(<\text{DF}_6>\)

(each \( t_i \) is a clock cycle, i.e. an atomic time unit)

An alternative organization

- \( t_1 \): \( \text{MAR} \leftarrow \text{PC} \)
- \( t_2 \): \( \text{MBR} \leftarrow \text{memory} \)
- \( t_3 \): \( \text{PC} \leftarrow \text{PC} + 1 \)
  \( \text{IR} \leftarrow \text{MBR} \)
Rules for micro-operation sequencing

- Proper precedence must be observed
  - MAR <- PC must precede MBR <- memory
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- memory & IR <- MBR must not be in same cycle
- Also: PC <- PC +1 involves addition
  - Depending on the kind of ALU may need additional micro-operations, hence it is better to have it in $t_2$
- Minimization of the number of micro-operations is an algorithmic problem on graphs
Sequence of micro-operations for direct addressing

- $t_1$: $\text{MAR} \leftarrow \text{IR}_{\text{address}} <DF_1>$
- $t_2$: $\text{MBR} \leftarrow \text{memory} <DF_2 \ DF_3 \ DF_4 \ DF_5>$
Sequence of micro-operations for register addressing

- $t_1$: Reg. $\leftarrow IR_{\text{register-address}} <DF_1>$
Sequence of micro-operations for register indirect addressing

- $t_1$: $\text{MAR} \leftarrow (\text{IR}_{\text{register-address}}) \ <\text{DF}_1\ \text{DF}_2 >$
- $t_2$: $\text{MBR} \leftarrow \text{memory} \ <\text{DF}_3\ \text{DF}_4\ \text{DF}_5\ \text{DF}_6 >$

**Diagram Notes:**
- MAR = Memory Address Register
- MBR = Memory Buffer Register
- IR = Instruction Register
- PC = Program Counter

Rev. 3.2 (2009-10) by Enrico Nardelli
Sequence of micro-operations for indirect addressing

- $t_1$: MAR <- IR\textsubscript{address} <DF\textsubscript{1} >
- $t_2$: MBR <- memory <DF\textsubscript{2} DF\textsubscript{3} DF\textsubscript{4} DF\textsubscript{5} >
- $t_3$: MAR <- MBR <DF\textsubscript{6} >
- $t_4$: MBR <- memory <DF\textsubscript{7} DF\textsubscript{8} DF\textsubscript{9} DF\textsubscript{10} >
Sequence of micro-operations for relative addressing

- $t_1$: $\text{MAR} \leftarrow \text{IR}_{\text{address}} + \text{PC}$  
  $<DF_1 \ DF_2 \ DF_3 \ DF_4>$

- $t_2$: $\text{MBR} \leftarrow \text{memory}$  
  $<DF_5 \ DF_6 \ DF_7 \ DF_8>$

\[ 
\text{MAR} = \text{Memory Address Register} \\
\text{MBR} = \text{Memory Buffer Register} \\
\text{IR} = \text{Instruction Register} \\
\text{PC} = \text{Program Counter} 
\]
Sequence of micro-operations for base and indexed addressing

- $t_1$: $\text{MAR} \leftarrow (\text{IR}_{\text{register-address}}) + \text{IR}_{\text{address}}$
  
  $<\text{DF}_1 \ \text{DF}_2 \ \text{DF}_3 \ \text{DF}_4 \ \text{DF}_5 >$

- $t_2$: $\text{MBR} \leftarrow \text{memory} <\text{DF}_6 \ \text{DF}_7 \ \text{DF}_8 \ \text{DF}_9 >$

MAR = Memory Address Register
MBR = Memory Buffer Register
IR = Instruction Register
PC = Program Counter
Sequence of micro-operations for combination of displacement and indirect addressing

- Try them yourself!
Sequence of micro-operations for interrupt handling

- **t_1**: MBR <- PC  
  MAR <- Stack-Pointer
  <DF_{1a}>  
  <DF_{1b}>

- **t_2**: memory <- MBR  
  Stack-Pointer <- Stack-Pointer + 1  
  <DF_{1c} DF_{1d} DF_{1e}>

- **t_3**: MAR <- Interrupt_Address
  <DF_{2a}>

- **t_4**: MBR <- memory
  <DF_{2b} DF_{2c} DF_{2d} DF_{2e}>

- **t_5**: PC <- MBR
  <DF_{2f}>

- NOTE: We assume Interrupt_Address is a fixed location known by the Control Unit
Micro-operation sequencing for the execution phase (1)

- Different for each instruction

- SUM X – sum the contents of memory cell X and Accumulator and store back the result in cell X

  Assuming that after the operand fetch phase content of cell at address X is in MBR:

  \[
  \begin{align*}
  t_1: & \quad \text{ALU} \leftarrow \text{AC} + \text{MBR} \\
  t_2: & \quad \text{AC} \leftarrow \text{ALU} \\
  t_3: & \quad \text{MBR} \leftarrow \text{AC}; \quad \text{MAR} \leftarrow \text{IR}_{\text{address}} \\
  t_4: & \quad \text{memory} \leftarrow \text{MBR}
  \end{align*}
  \]
Micro-operation sequencing for the execution phase (2)

• ISZ X - increment memory cell X and if it’s zero skip the next instruction
  Assuming that content of cell at address X is in MBR after the operand fetch phase:
  ▪ \( t_1: \) ALU \(<-\) MBR + 1
  ▪ \( t_2: \) MBR \(<-\) ALU
  ▪ \( t_3: \) memory \(<-\) MBR
    \(\text{IF MBR} = = 0 \text{ THEN PC} <- \text{PC} + 1\)

• Note:
  ▪ IF-THEN is a single micro-operation
Micro-operation sequencing for the execution phase (3)

- CALL X - Save in stack the return address and jump to address X
  - \( t_1: \) \( MBR \leftarrow PC \)
    \[
    \text{MAR} \leftarrow \text{SP}
    \]
  - \( t_2: \) memory \( \leftarrow \) MBR
    \[
    \text{ALU} \leftarrow \text{SP} + 1
    \]
    \[
    \text{PC} \leftarrow \text{IR}_{\text{address}}
    \]
  - \( t_3: \) SP \( \leftarrow \) ALU
A simplified flow diagram for the execution of instruction cycle

ICC = Instruction Cycle Code

- 11 (Interrupt)
  - Opcode?
    - Execute interrupt handling micro-ops
      - ICC = 00
        - Interrupt enabled?
          - ICC = 11
          - ICC = 00

- 10 (Execution)
  - ICC = 10
  - Execute indirect addressing micro-ops
    - ICC = 00
    - ICC = 10

- 01 (Indirect Addressing)
  - Execute instruction fetch micro-ops
    - ICC = 01
Functions of Control Unit

- Sequencing
  - Causing the CPU to step through a series of micro-operations
- Execution
  - Causing the execution of each micro-op
- ALL THESE ACTIONS are performed by means of Control Signals
A simplified data flow diagram of a Control Unit
Control Unit’s Input Signals

- **Clock**
  - One micro-op (or set of parallel ops) per clock cycle
  - Different signals are needed for different steps
- **Instruction register**
  - Op-code for current instruction
  - Determines which micro-instructions are performed
- **Flags**
  - State of CPU
  - Results of previous operations
- **Control Bus**
  - Interrupts
  - Acknowledgments
Control Unit’s Output Signals

- To other CPU components
  - For data movement
  - To activate specific functions
- To the Control Bus
  - To control memory
  - To control I/O modules
- Output signals from the control unit (i.e., Control Signals) make all micro-operations happen
Simplified schema of a CPU: connections and control gates

Control Gate $x$: The flow of data is enabled when the Control Signal $x$ is enabled.

Rev. 3.2 (2009-10) by Enrico Nardelli
Example of Control Signal Sequence – Instruction Fetch (1)

- \( t_1 \): MAR \( \leftarrow \) PC
  - Control unit (CU) activates signal C2 to open gate from PC to MAR
- \( t_{2-1} \): MBR \( \leftarrow \) memory
  - CU activates C0 to open gate from MAR to address bus
  - CU activates the \textit{memory read} control signal (CR - not shown) to the memory
  - CU activates C5 to open gate from data bus to MBR
Example of Control Signal Sequence – Instruction Fetch (2)

- $t_{2-2}$: PC <- PC +1
  - In the considered CPU’s internal schema, ALU’s output is not directly connected to PC but only to AC. Therefore this micro-operation has to be split in two subsequent time units.
  - What would happen if we had ALU’s output directly connected to PC?

- $t_3$: IR <- MBR
  - CU activates C4 to open gate from MBR to IR
Example of Control Signal Sequence – Instruction Fetch (3)

• Splitting the increment of Program Counter
  ▪ ALU is a fast combinational circuit whose inputs and output are **not** buffered
  ▪ ALU has a specific control signal CA for unitary increment without a second input

• \( t_{2-2} \): \( PC \leftarrow PC + 1 \)
  ▪ \( t_{2-2-1} \): \( ALU \leftarrow PC \)  \( \text{CU activates C14 from PC to ALU} \)
  ▪ \( ALU \) \( \text{CU act. control signal CA (not shown) for ALU} \)
  ▪ \( AC \leftarrow ALU \)  \( \text{CU activates C9 from ALU to PC} \)
  ▪ \( t_{2-2-2} \): \( PC \leftarrow AC \)  \( \text{CU activates C15 from AC to PC} \)
Example of Control Signal Sequence – Instruction Fetch (4)

- Optimization
  - $t_{2-1}$ and $t_{2-2-1}$ can be executed together
  - $t_{2-2-2}$ and $t_3$ can be executed together

- New organization
  - $t_1$: MAR $\leftarrow$ PC C2
  - $t_2$: MBR $\leftarrow$ memory C0 CR C5
    - ALU $\leftarrow$ PC C14
    - increment ALU CA
    - AC $\leftarrow$ ALU C9
  - $t_3$: PC $\leftarrow$ AC C15
  - IR $\leftarrow$ MBR C4
Example of Control Signal Sequence - Direct Addressing

- Direct addressing is executed right after instruction fetch in our simplified flow diagram for the execution of an instruction cycle. Hence:
  - $t_4$: $\text{MAR} \leftarrow \text{IR}_{\text{address}}$
    - CU activates C16 to open gate from IR to MAR
  - $t_5$: $\text{MBR} \leftarrow \text{memory}$
    - CU act. C0 to open gate from MAR to address bus
    - CU act. the $\text{memory read}$ control signal CR
    - CU act. C5 to open gate from data bus to MBR
Example of Control Signal Sequence - Indirect Addressing

• \( t_1 \): \text{MAR} \leftarrow \text{IR}_{\text{address}}
  \begin{itemize}
    \item CU activates C16 to open gate from IR to MAR
  \end{itemize}

• \( t_2 \): \text{MBR} \leftarrow \text{memory}
  \begin{itemize}
    \item CU act. C0 to open gate from MAR to address bus
    \item CU act. the \textit{memory read} control signal CR
    \item CU act. C5 to open gate from data bus to MBR
  \end{itemize}

• \( t_3 \): \text{MAR} \leftarrow \text{MBR}
  \begin{itemize}
    \item CU activates C8 to open gate from MBR to MAR
  \end{itemize}

• \( t_4 \): \text{MBR} \leftarrow \text{memory}
  \begin{itemize}
    \item CU activates C0, CR and C5 as above
  \end{itemize}
Example of Control Signal Sequence – Execution: SUM

- $t_1$: ALU <- AC + MBR
  - CU activates C6 to open gate from MBR to ALU
  - CU activates C7 to open gate from AC to ALU
  - CU activates for the ALU the *sum* control signal CS

- $t_2$: AC <- ALU
  - CU activates C9 to open gate from ALU to AC

- $t_2$: MBR <- AC; MAR <- IR$_{\text{address}}$
  - CU activates C11 to open gate from AC to MBR
  - CU activates C16 to open gate from IR to MAR

- $t_2$: memory <- MBR
  - CU activates C0 to open gate from MAR to address bus
  - CU activates C12 to open gate from MBR to data bus
  - CU activates the *memory write* control signal CW

- NOTE: Now ALU’s output needs to be “buffered”: this means that its output lines are not directly coming from the internal combinational circuits but from a registry (*buffer*) that receives combinational circuits outputs and store them for a subsequent reading.
Limitations

• The simplified internal schema of a CPU does not show registers, hence we cannot show
  - Register addressing
  - Register indirect addressing
  - Base addressing
  - Indexed addressing
  - Combination of displacement and indirect addressing

• Try adding to the simplified schema one or more of the above addressing modalities and derive the required micro-operations!
Internal Organization of CPU

- Usually a single internal bus
  - less complex than having direct data paths between registers and ALU
- Control gates control movement of data onto and off the internal bus
- Control signals control also data transfer to and from external systems bus
- Temporary registers (i.e., buffers) in input to ALU are now needed for proper operation of ALU
- After Appendix B try yourself deriving the control signal sequences for a single internal bus CPU!
How to implement the instruction cycle in hardware?

ICC = Instruction Cycle Code

11 (Interrupt) → Execute interrupt handling micro-ops

10 (Execution) → Opcode?

00 (Fetch) → Execute instruction fetch micro-ops

01 (Indirect Addressing) → Indirect addressing?

Yes

ICC = 01 → Execute direct addressing micro-ops

No

Interrupt enabled?

 ICC = 00

 Execute micro-ops for the given opcode

 ICC = 11

 ICC = 00

 ICC = 10

 ICC = 00

 ICC = 10

 ICC = 10

ICC = 00
Hardwired Implementation (1)

- Consider the control unit as a combinational circuit
  - Outputs of the circuit are the control signals
  - Inputs of the circuit are:
    - ICC register bits (note that ICC is not visualized in the CPU’s simplified internal schema)
    - Status flags, including interrupt-enable/disable (T)
    - Direct/indirect address bit (D)
    - Decoded opcodes ($OC_n$)
    - Clocks ($t_n$)
  - For each configuration of inputs produce a proper output
  - That is, the activation of a given control signal $C_n$ has to happen when (condition $A$ is true) OR (condition $B$ is true) OR ...
Data flow diagram of the Control Unit for the hardwired implement of the CPU’s simplified schema.
Hardwired Implementation (2)

- Control signals P and Q code ICC: then fetch (instruction +
direct addr.) is coded by P’Q’, indirect by P’Q, execute by
PQ’, and interrupt by PQ
- Direct addressing is coded by D (D’ = indirect)
- Decoded opcodes provide are further control signals
- Interrupt enable is coded by T (T’ = interrupt disabled)
- Each clock t_n is a control signal
- Boolean expression activating C5 in the simplified schema of
a CPU (see slides 26, 27, 28 and 13):
\[ C5 = P’Q’(t_2 + Dt_5) + P’Q(t_2 + t_4) + PQ’B + PQ t_4 \]
  - where B is the boolean expression representing, for all opcodes
    activating C5, all micro-operations actually activating it
  - example: if C5 is activated only by opcode 3 during t_2 and t_4 and by
    opcode 7 during t_3 and t_4 then B is: OC3(t_2+t_4)+OC7(t_3+t_4)
Hardwired Implementation (3)

- **Updating ICC:**
  - Assume no micro-procedure requires more than 5 time units
  - Update P and Q using clock $t_6$, current values of P and Q, interrupt enabled flag (T), direct address signal (D)
  - At the end of direct addressing branch of fetch phase (00)
    - $P = P'Q' \ t_6 \ D$
    - $Q' = P'Q' \ t_6 \ D$
  - At the end of indirect addressing branch of fetch phase (00)
    - $P' = P'Q' \ t_6 \ D'$
    - $Q = P'Q' \ t_6 \ D'$
  - At the end of indirect addressing phase (01)
    - $P = P'Q \ t_6$
    - $Q' = P'Q \ t_6$
  - At the end of interrupt enabled branch of execution phase (10)
    - $P = PQ' \ t_6 \ T$
    - $Q = PQ' \ t_6 \ T$
  - At the end of interrupt disabled branch of execution phase (10)
    - $P' = PQ' \ t_6 \ T'$
    - $Q' = PQ' \ t_6 \ T'$
  - At the end of interrupt phase (11)
    - $P' = PQ \ t_6$
    - $Q' = PQ \ t_6$
Do we forget anything?

- Discarding $t_6$ (present in all terms) we have

\[
P = P'Q' D + P'Q + PQ' T \quad \text{and} \quad P' = P'Q' D' + PQ' T' + PQ \\
Q = P'Q' D' + PQ' T \quad \text{and} \quad Q' = P'Q' D + P'Q + PQ' T' + PQ
\]

<table>
<thead>
<tr>
<th>$P$</th>
<th>$P'Q'$</th>
<th>$P'Q$</th>
<th>$PQ$</th>
<th>$PQ'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$P'Q'$</th>
<th>$P'Q$</th>
<th>$PQ$</th>
<th>$PQ'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Problems with the Hardwired Implementation

• Complex sequencing & micro-operation logic
• Difficult to design and test
• Inflexible design
• Difficult to add new instruction