What is a program?

- A sequence of instructions (steps)
  - an instruction
    - a binary number
    - Consisting of two parts
      - Opcode: what to do
      - Operand: what data should be used
    - Ex:
      
      | Opcode | Operand |
      |--------|--------|

- For each instruction (step), an arithmetic or logical operation is done

- For each operation, a different set of control signals is needed
Function of Control Unit

- Each operation is associated with a unique opcode
  - e.g. ADD, MOVE
- The hardware interprets the opcode and issues the control signals
- We have a computer!

Computer Components: Top Level View

Instruction Cycle

- Two steps:
  - Fetch
  - Execute
**Fetch Cycle**
- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

**Execute Cycle**
- Processor-memory
  - Data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- Combination of above

**Example of Program Execution**

<table>
<thead>
<tr>
<th>Step</th>
<th>Memory</th>
<th>CPU Registers</th>
<th>PC</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0b0100</td>
<td>0b000 0 0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0b0100</td>
<td>0b000 0 0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0b0100</td>
<td>0b000 0 0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0b0100</td>
<td>0b000 0 0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0b0100</td>
<td>0b000 0 0000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

0b0001 = 0x1
Load AC from memory
0b0010 = 0x2
Store AC to memory
0b0101 = 0x5
Add to AC from memory
0b0101 = 0x5
Add to AC from memory

PC: program counter
AC: accumulator
IR: instruction register
Outline
- Processor structure and instruction cycles
- Pipelining basic
- Interrupt
- Buses

What is Pipeline?
- Laundry Example
  - Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
  - Washer takes 30 minutes
  - Dryer takes 40 minutes
  - "Folder" takes 20 minutes
Sequential Laundry

6 PM  7  8  9  10  11 Midnight

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>40</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>20</td>
</tr>
</tbody>
</table>

Task Order

Total = 6 hours

Pipelined Laundry: Start ASAP

6 PM  7  8  9  10  11 Midnight

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Task Order

Total: 3.5 hours

Pipelining

- Overlap the executions of multiple instructions
  - Fetch instruction (FI)
  - Decode instruction (DI)
  - Calculate operands (i.e. EAs) (CO)
  - Fetch operands (FO)
  - Execute instructions (EI)
  - Write result (WO)
Timing Diagram for Instruction Pipeline Operation

Pipeline Performance

- Speedup factor
  - Let
    - each stage take one clock cycle
    - an instruction takes k stages
  - $T_1$: without pipeline, a program with n instruction will take $nk$ cycles
  - $T_2$: with k stage pipeline, only the first instruction takes $k$ cycles and the rest of them takes only one cycles.
  - Therefore
    \[
    S_k = \frac{T_1}{T_k} = \frac{nk}{k + (n - 1)} \approx k
    \]
    - When $n \to \infty$

Pipeline Hazards

- Pipeline, or some portion of pipeline, must stall
- Also called pipeline bubble
- Types of hazards
  - Resource
    - Two (or more) instructions in pipeline need same resource
    - Ex: both instructions need multipliers
  - Data
    - Data dependency
    - Ex: one instruction needs the results from the previous one that has not been available yet
  - Control
    - The next instruction is not at PC+1
    - Ex: Branch, procedure call and return
**Pipeline Principles**

- Pipelining doesn’t help latency of a single task, it helps throughput of entire workload.
- The pipeline rate limited by slowest pipeline stage.
- Multiple tasks operating simultaneously.
- Potential speedup = Number of pipe stages.
- Sometimes, pipeline has to be stalled due to the pipeline hazard.

**Example**

- Assume an unpipelined computer
  - 1 ns clock cycle
  - Branches and ALU ops take 4 cycles
  - Memory ops take 5 cycles
  - Instruction mix:
    - 20% branch, 40% ALU, 40% mem
- Enhanced with a 5 stage pipeline
  - .2 ns overhead for clock skew and register delay
  - Assume that each stage takes 1 clock cycle
- Speedup?

**Outline**

- Processor structure and instruction cycles
- Pipelining basic
- Interrupt
- Buses
### Interrupt Mechanism

- **What**
  - Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
  - A way to improve processing efficiency

- **Why**
  - Processor usually much faster than the I/O devices
  - Waste of processor to wait I/O operations done

### Interrupt Mechanism

- **How**
  - Interrupt request from I/O module
  - Processor acknowledges the request
  - Suspending current program
  - Save the context
    - Values of PC, registers, etc
  - Go to service routine (interrupt handler)
  - Restore the context
  - Resume the program

### Program Flow Control

![Program Flow Control Diagram](image-url)
Program Timing
Short I/O Wait

1. Start
2. Process
3. I/O operation
4. Process
5. End

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 (ex: registers, PC, etc)
  — Set PC to start address of interrupt handler routine
  — Process interrupt
  — Restore context and continue interrupted program

Program Timing
Long I/O Wait

1. Start
2. Process
3. I/O operation
4. Process
5. I/O operation
6. Process
7. I/O operation
8. Process
9. I/O operation
10. Process
11. End

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 (ex: registers, PC, etc)
  — Set PC to start address of interrupt handler routine
  — Process interrupt
  — Restore context and continue interrupted program
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

![Diagram of Sequential Multiple Interrupts]

Multiple Interrupts – Nested

![Diagram of Nested Multiple Interrupts]

Efficiency

- Ex:
  - User program with 1,000,000 instructions
  - One print job (I/O operation) requiring 5 seconds
  - Processor speed at 1Ghz, 1 cycle per instruction
  - Interrupt handler 10,000 instructions
  - Ignore interrupt overhead (context saving, etc)
  - Speedup w/o interrupt?
Outline

- Processor structure and instruction cycles
- Pipelining basic
- Interrupt
- Buses

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

Bus Interconnection Scheme
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 Hierarchy

- Lots of devices on one single 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

High Performance Bus

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
**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 or Distributed Arbitration**
- Centralised
  - Single hardware device controlling bus access
    - Bus Controller
    - Arbiter
  - May be part of CPU or separate
- Distributed
  - Each module may claim the bus
  - Control logic on all modules

**Summary**
- Processor structure and instruction cycles
  - High level view of processor, program execution, instruction cycles
- Pipelining basic
  - Overlap execution of multiple instruction
  - Pipeline principles
- Interrupt
  - What/why/how
  - Instruction cycles with interrupt
  - Multiple interrupts
- Buses
  - What
  - Data/address/control
  - Dedicated/multiplexed, centralized/distributed