Performance & Power

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Performance
Elements of CPU Performance

- Cycle time
- CPU pipeline
- Memory systems
Pipelining

▪ Several instructions are executed simultaneously at different stages of completion
▪ Various conditions can cause pipeline bubbles that reduce utilization:
  • branches
  • memory system delays
  • dependencies among instructions
Performance Measures

▪ Latency
  • Time it takes for an instruction to get through the pipeline

▪ Throughput
  • The number of instructions executed per time period

▪ Pipelining increases throughput without reducing latency
ARM7 Pipeline

- ARM7 has 3-stage pipeline:
  - **Fetch** instruction from memory
  - **Decode** opcode and operands
  - **Execute**
ARM7 Pipeline Execution

```
add r0, r1, #5
sub r2, r3, r6
cmp r2, #3
```

Time:

1  2  3
Pipeline Stalls

- If every step cannot be completed in the same amount of time, pipeline stalls.

- Bubbles introduced by stall increase latency, reduce throughput
ARM Multi-cycle LDMIA

ldmia  r0,{r2,r3}
sub    r2,r3,r6
cmp    r2,#3
Control Stalls

- Branches often introduce stalls (branch penalty)
  - Stall time may depend on whether branch is taken
- May have to squash instructions that already started executing
- Don’t know what to fetch until condition is evaluated
ARM Pipelined Branch

bne  foo
sub  r2, r3, r6
foo:
    add r0, r1, r2
Delayed Branch

▪ To increase pipeline efficiency, delayed branch mechanism requires \( n \) instructions after branch always executed whether branch is executed or not.

▪ Use NOP (No Operation) instruction if there are not enough instructions to fill the delayed branch window.
Example: FIR Filter (1)

- Execution time of FIR filter?
- Only branch in loop test may take more than one cycle
  - BLT loop takes 1 cycle best case, 3 cycles worst case

```c
for (i = 0, f = 0; i < N; i++)
    f = f + c[i] * x[i];
```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR</td>
<td>R4, [R3, R8] ; R4 = c[i]</td>
</tr>
<tr>
<td>LDR</td>
<td>R6, [R5, R8] ; R6 = x[i]</td>
</tr>
<tr>
<td>MUL</td>
<td>R4, R4, R6 ; R4 = c[i]*x[i]</td>
</tr>
<tr>
<td>ADD</td>
<td>R2, R2, R4 ; f += c[i]*x[i]</td>
</tr>
<tr>
<td>ADD</td>
<td>R8, R8, #4 ; R8 += 4</td>
</tr>
<tr>
<td>ADD</td>
<td>Ro, Ro, #1 ; i += 1</td>
</tr>
<tr>
<td>CMP</td>
<td>Ro, R1 ; if (i &lt; N)</td>
</tr>
<tr>
<td>BLT</td>
<td>loop ; Continue</td>
</tr>
</tbody>
</table>
Example: FIR Filter (2)

<table>
<thead>
<tr>
<th>Block</th>
<th>Variable</th>
<th># instructions</th>
<th># cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>$t_{\text{init}}$</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Body</td>
<td>$t_{\text{body}}$</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Update</td>
<td>$t_{\text{update}}$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Test</td>
<td>$t_{\text{test}}$</td>
<td>2</td>
<td>[2, 4]</td>
</tr>
</tbody>
</table>

$$t_{\text{loop}} = t_{\text{init}} + N(t_{\text{body}} + t_{\text{update}}) + (N-1) t_{\text{test, worst}} + t_{\text{test, best}}$$

Loop test succeeds is worst case
Loop test fails is best case
Memory System Performance

- Caches introduce indeterminacy in execution time
- Cache performance depends on order of execution
- **Cache miss penalty**: added time due to a cache miss
Type of Cache Misses

- **Compulsory miss**
  - Location has not been referenced before

- **Conflict miss**
  - Two locations are fighting for the same block

- **Capacity miss**
  - Working set is too large
Scratch pad memories

- Alternative to cache:
  - Software determines what is stored in scratch pad.
- Provides predictable behavior at the cost of software control.
- In some CPUs/DSPs, cache can be configured as scratch pad memory.
Power Consumption
CPU Power Consumption

- Most modern CPUs are designed with power consumption in mind to some degree.

- Energy = Power x Time
  - Heat depends on power consumption
  - Battery life depends on energy consumption
CMOS Power Consumption

- Voltage drops
  - Power consumption proportional to $V^2$
- Toggling
  - More activity means more power
- Leakage
  - Basic circuit characteristics
  - Can be eliminated by disconnecting power
CPU Power-saving Strategies

- Reduce power supply voltage
- Run at lower clock frequency
- Disable function units with control signals when not in use
- Disconnect parts from power supply when not in use
Power Management Styles

- **Static power management**
  - Does not depend on CPU activity
  - e.g., user-activated power-down mode

- **Dynamic power management**
  - Based on CPU activity
  - e.g., disabling off function units
StrongARM SA-1100

- Processor takes two supplies:
  - VDD is main 3.3V supply
  - VDDX is 1.5V

- Three power modes
  - Run: normal operation
  - Idle: stops CPU clock, with logic still powered
  - Sleep: shuts off most of chip activity; 3 steps, each about 30μs; wakeup takes > 10 ms
SA-1100 Power State Machine

$P_{\text{run}} = 400 \text{ mW}$

$P_{\text{idle}} = 50 \text{ mW}$

$P_{\text{sleep}} = 0.16 \text{ mW}$