Elements of CPU performance

- Cycle time.
- CPU pipeline.
- Superscalar design.
- Memory system.

\[ T_{exec} = \left( \frac{\text{instructions}}{\text{program}} \right) \left( \frac{\text{cycles}}{\text{instruction}} \right) \left( \frac{\text{seconds}}{\text{cycle}} \right) \]
ARM7TDM CPU Core
ARM Cortex A-9 MPcore
### Pipelining

- Several instructions are executed simultaneously at different stages of completion.

<table>
<thead>
<tr>
<th>I1</th>
<th>Fetch</th>
<th>Decode</th>
<th>Execute</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
</tr>
<tr>
<td>I4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I5</td>
<td>Fetch</td>
<td>Decode</td>
<td>Execute</td>
<td>Write</td>
</tr>
</tbody>
</table>

- Various conditions can cause **pipeline bubbles** that reduce utilization:
  - branches;
  - memory system delays;
  - etc.
ARM pipeline execution

ARM7 has 3-stage pipes:
- fetch instruction from memory;
- decode opcode and operands;
- execute.

```
add r0, r1, #5
```

```
sub r2, r3, r6
```

```
cmp r2, #3
```
Pipeline changes for ARM9TDMI
ARM10 and ARM11 pipelines

ARM10

- Branch Prediction
- ARM or Thumb Instruction Decode
- Reg Read
- Shift + ALU
- Memory Access
- Reg Write

 FETCH  ISSUE  DECODE  EXECUTE  MEMORY  WRITE

ARM11

(Superscalar design)

- Fetch 1
- Fetch 2
- Decode
- Issue
- MAC 1
- MAC 2
- MAC 3

- Write back
- Address
- Data Cache 1
- Data Cache 2
Performance measures

- **Latency**: time it takes for an instruction to get through the pipeline.
- **Throughput**: number of instructions executed per time period.

Pipelining increases throughput without reducing latency.

Assume a program with $N$, $K$-stage instructions

- Without pipeline: $T_{exec} = N \times K$
- With $K$-stage pipeline: $T_{exec} = K + (N-1)$
  
  - $K$ cycles for 1st instruction
  - 1 cycle to complete each additional instruction

$$\text{Speedup} = \frac{N \times K}{K + (N-1)}$$

For large $N$

$$\text{Speedup} \approx K$$

This assumes no pipeline stalls.
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 instruction

ldmia
r0, {r2, r3}

sub
r2, r3, r6

cmp
r2, #3

time

fetch decode ex ld r2 ex ld r3

fetch decode ex sub

fetch decode ex cmp
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
  fetch  decode  ex bne  ex bne  ex bne

sub  r2, r3, r6
  fetch  decode

foo add  r0, r1, r2
  fetch  decode  ex add
```

Diagram showing the pipeline stages for ARM instructions.
Delayed branch

- To increase pipeline efficiency, delayed branch mechanism requires $n$ instructions after branch always executed whether branch is executed or not.

- SHARC supports delayed and non-delayed branches.
  - Specified by bit in branch instruction.
  - 2 instruction branch delay slot.
Example: ARM execution time

- Determine execution time of FIR filter:
  ```c
  for (i=0; i<N; i++)
      f = f + c[i]*x[i];
  ```

- Only branch in loop test may take more than one cycle.
  - BLT loop takes 1 cycle best case, 3 worst case.
FIR filter ARM code

; loop initiation code
MOV r0,#0 ; use r0 for i, set to 0
MOV r8,#0 ; use a separate index for arrays
ADR r2,N ; get address for N
LDR r1,[r2] ; get value of N
MOV r2,#0 ; use r2 for f, set to 0
ADR r3,c ; load r3 with address of base of c
ADR r5,x ; load r5 with address of base of x

; loop body
loop
LDR r4,[r3,r8] ; get value of c[i]
LDR r6,[r5,r8] ; get value of x[i]
MUL r4,r4,r6 ; compute c[i]*x[i]
ADD r2,r2,r4 ; add into running sum
ADD r8,r8,#4 ; add one to array index
ADD r0,r0,#1 ; add 1 to i
CMP r0,r1
BLT loop ; if i < N, continue loop

loopend ...
FIR filter performance by block

<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
FIR performance on ARM

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

\[ \begin{array}{c c c c c c}
7 & 4 & 2 & 4 & 2 \\
\end{array} \]

\( N = \# \text{ times loop executed} \)

\[ t_{\text{loop}} = 5 + (N \times 10)\text{cycles} \]
Superscalar execution

- Superscalar processor can execute several instructions per cycle.
  - Uses multiple pipelined data paths.
- Programs execute faster, but it is harder to determine how much faster.
- Multicore module has multiple processors, each executing separate program “threads”
Data dependencies

- Execution time depends on operands, not just opcode.
- Superscalar CPU checks data dependencies dynamically:

```
add r2, r0, r1
add r3, r2, r5
```

Data dependency graph:

```
add r2, r0, r1
add r3, r2, r5
```
C55x pipeline

- C55x has 7-stage pipe:
  - fetch;
  - decode;
  - address: computes data/branch addresses;
  - access 1: reads data;
  - access 2: finishes data read;
  - Read stage: puts operands on internal busses;
  - execute.
C55x organization

- 3 data read busses
- 3 data read address busses
- Program address bus
- Program read bus
- 2 data write busses
- 2 data write address busses

Instruction unit
Program flow unit
Address unit
Data unit

B bus

16
24
24
16
24
32

Instruction fetch
Data read from memory
Single operand read
C, D busses
Dual operand read
B bus
Dual-multiply coefficient
Writes
C55x pipeline hazards

- Processor structure:
  - Three computation units.
  - 14 operators.
- Can perform two operations per instruction.
- Some combinations of operators are not legal.
C55x hazards

- A-unit ALU/A-unit ALU.
- A-unit swap/A-unit swap.
- D-unit ALU, shifter, MAC/D-unit ALU, shifter, MAC
- D-unit shifter/D-unit shift, store
- D-unit shift, store/D-unit shift, store
- D-unit swap/D-unit swap
- P-unit control/P-unit control