ECE 571 – Advanced Microprocessor-Based Design Lecture 5

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 September 2024

Announcements

- Homework #1 was due
- Homework #2 will be posted. A reading on measurement bias, with some short answer questions.

On simple processors often take multiple cycles to finish

- 6502, 1MHz, instructions roughly 1 cycle per mem access
  - $69,$0A - adc #10 - 2 cycles (add immediate)
  - $65,$10 - adc $10 - 3 cycles (add zero page)
  - $70,$34,$12 - adc $1234,X - 4+ cycles (add absolute indexed)

    can take extra cycle if wraps page (carry from add)
- Racing the beam and Vapor Lock

IPC Metric

- Instructions per Cycle
- Higher is better
- Inverse of CPI (cycles per instruction)

How can we increase IPC?

Advanced CPUs

Some sample code

```
int i;
int x[128];

for(i=0;i<128;i++) {
          x[i]=0;
}
```

How do you convert this to something the CPU understands?

Roughly Equivalent Assembler

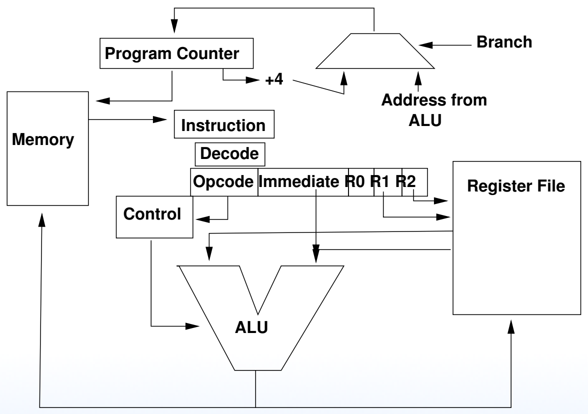
```
    mov r0,#0          ; i=0
loop:
    ldr r1,=x          ; point r1 to x array
    lsl r2,r0,#2       ; r2=i*4
    mov r3,#0          ; want to write 0 to x[i]
    str r3,[r1,r2]     ; x[i]=0
    add r0,r0,#1       ; i++
    cmp r0,#128        ; is i==128?
    bne loop           ; if not, keep looping

    ; Note: can do lots of code hoisting here

.bss
.lcomm  x,128*4
```

An aside: how could you optimize this code?

- Unroll the loop?
- Code hoisting (move the pointer load outside the loop)
- Use larger-sized writes (64-bit?)
- Use ARM barrel-shift addressing modes
- Crazy x86 instructions `rep stosb`

Simple CPUs

- Ran one instruction at a time.
- Could take one or multiple cycles (IPC 1.0 or less)
- Example – single instruction take 1-5 cycles?



IPC Metric

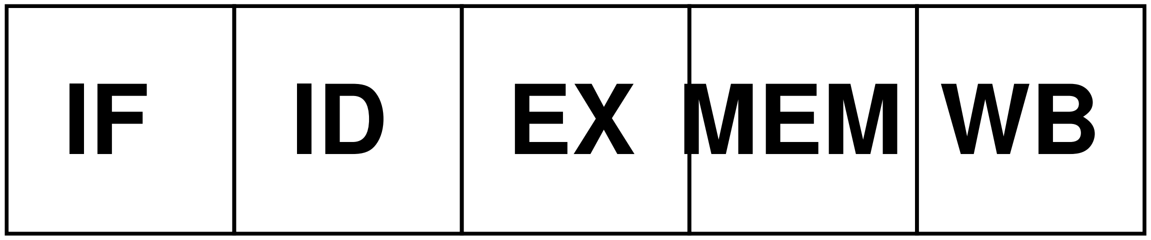
- Instructions per Cycle
- Higher is better
- Inverse of CPI (cycles per instruction)

How can we increase IPC?

- Simple CPU must have cycles as slow as slowest instruction
- What if we break instructions up to take multiple cycles?
- What if we could overlap them?

Pipelined CPUs

- 5-stage MIPS pipeline
- Have you ever used one? The first time I used UNIX: MIPS R3000 in an SGI Personal Iris 4D/35
- Original Playstation



Pipelined CPUs

- IF = Instruction Fetch, Update PC
  Fetch 32-bit instruction from L1-cache
- ID = Decode, Fetch Register Values
- EX = execute (ALU, maybe shifter, multiplier, divide)
  Memory address calculated
- MEM = Memory – if memory had to be accessed, happens now.
- WB = register values written back to the register file

Cycle 1

IF	`mov r0,#0`
ID	
EX	
MEM	
WB	

Cycle 2

IF	`ldr r1,=x`
ID	`mov r0,#0`
EX	
MEM	
WB	

Cycle 3

IF	`lsl r2,r0,#2`
ID	`ldr r1,=x`
EX	`mov r0,#0`
MEM	
WB	

Cycle 4

IF	`mov r3,#0`
ID	`lsl r2,r0,#2`
EX	`ldr r1,=x`
MEM	`mov r0,#0`
WB	

Cycle 5

IF	`str r3,[r1,r2]`
ID	`mov r3,#0`
EX	`lsl r2,r0,#2`
MEM	`ldr r1,=x`
WB	`mov r0,#0`

Benefits/Downside

- Latency higher (5 cycles) but average might be 1 cycle
- Why bother? Can you run the clock faster?
- From 2-stage to Pentium 4 31-stage

Data Hazards

Happen because instructions might depend on results from instructions ahead of them in the pipeline that haven't been written back yet.

- RAW – "true" dependency – problem. Bypassing?

EX	`add r1,r0,r0`
MEM	`ldr r0,[r1]`
WB	`mov r1,r3`

- WAR – "anti" dependency – not a problem if commit in order

- WAW – “output” dependency – not a problem as long as ordered
- RAR – not a problem

Structural Hazards

- CPU can’t just provide. Not enough multipliers for example

Control Hazards

- How quickly can we know outcome of a branch
- Branch prediction? Branch delay slot?

IF	`???`
ID	`beq ___`
EX	`cmp r0,r1`

Branch Prediction

- Predict (guess) if a branch is taken or not.
- What do we do if guess wrong? (have to have some way to cancel and start over)
- Modern predictors can be very good, greater than 99%
- Designs are complex and could fill an entire class

Memory Delay

- Memory/cache is slow
- Need to bubble / Memory Delay Slot

EX	`add r1,r1,r0`
MEM	`ldr r0,[r3]`

The Memory Wall

- Wulf and McKee
- Processors getting faster more quickly than memory
- Processors can spend large amounts of time waiting for memory to be available
- How do we hide this?

Caches

- Basic idea is that you have small, faster memories that are closer to the CPU and much faster
- Data from main memory is cached in these caches
- Data is automatically brought in as needed.
  Also can be pre-fetched, either explicitly by program or by the hardware guessing.
- What are the downsides of pre-fetching?
- Modern systems often have multiple levels of cache. Usually a small (32k or so each) L1 instruction and data,

a larger (128k?) shared L2, then L3 and even L4.

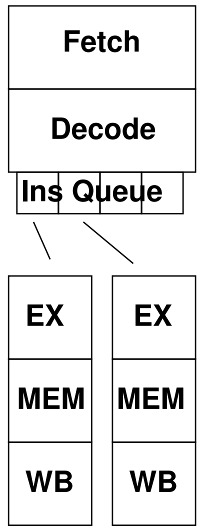
- Modern systems also might share caches between processors, more on that later
- Again, could teach a whole class on caches

Exploiting Parallelism

- How can we take advantage of parallelism in the control stream?
- Can we execute more than one instruction at a time?

Multi-Issue (Super-Scalar)

- Decode up to X instructions at a time, and if no dependencies issue at same time.
- Types
  - Static Multi-Issue – at compile time, VLIW
  - Dynamic
- Dual issue example. Can have theoretical IPC of 2.0
- Can have unequal pipelines.


Fetch
Decode
Ins Queue
EX
MEM
WB
EX
MEM
WB

Register Renaming

- Loop unrolling
- If only a “name” dependence
- Architectural register doesn’t have to be updated until written to
- Once written to it is essentially a separate register despite the same name

```
ldr r1,[1024]    ; ldr   r100,[1024]
add r1,#5        ; add   r100,#5
str r1,[2048]    ; str   r100,[2048]
ldr r1,[1025]    ; ldr   r101,[1025]
add r1,#5        ; add   r101,#5
str r1,[2049]    ; str   r101,[2049]
```