High Performance Computer Architecture

Madison Hanberry

July 30, 2019

Contents

(12 -13) Quiz: Memory Ordering

Lesson 2: Introduction

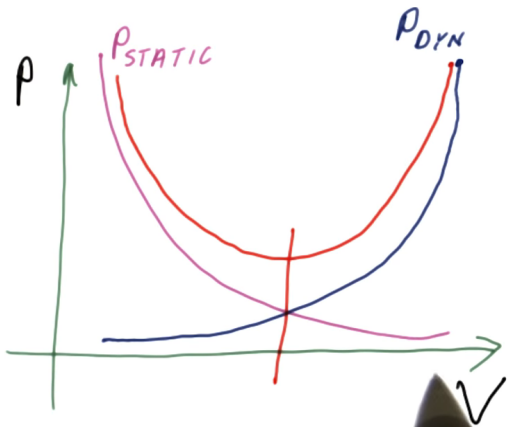
(12) Active Power

$$P = \frac{1}{2} \times v^2 \times f \times \alpha \tag{1}$$

C: Capacitance ($\sim$ Chip area)
v^2: Power supply voltage
f: Frequency
α: Activity factor (% of transistors active in any given clock cycle)

(13) Static Power

Decreasing *dynamic* power ($v \downarrow$) increases *static* power (increases power leakage).



(14) Quiz: Active Power

$$\begin{aligned} P &= 30W \ at \ 1.0v, 2 \ GHz \\ 30 &= \frac{1}{2} C \times (1.0)^2 \times 2\alpha \\ 30 &= C\alpha \end{aligned} \tag{2}$$

(i) $0.9v, 1.8\ GHz$

$$\begin{aligned} P &= \frac{1}{2}(30)(0.9)^2 \times 1.8 \\ P &= 21.87 \end{aligned} \tag{3}$$

(ii) $1.5v, 3\ GHz$

$$\begin{aligned} P &= \frac{1}{2}(30)(1.5)^2 \times 3 \\ P &= 101.25 \end{aligned} \tag{4}$$

(16) Fabrication Yield

$$Yield = \frac{Working\ Chips}{Chips\ on\ Wafer} \tag{5}$$

Yield decreases as Chip size decreases relative to the wafer size.

(17) Fabrication Cost 2

$$Chip\ Cost = \frac{Wafer\ Cost}{\#\ of\ Good\ Chips/Wafer} \tag{6}$$

Lesson 3: Metrics and Evaluation

(2) Performance

- Latency (start → Done)
- Throughput (# / second)

$$Throughput = \frac{1}{Latency} \tag{7}$$

(3) Quiz: Latency and Throughput

- 2 servers
- A server takes 1 msec to process an order

$$\begin{aligned} Throughput &= 2000\ orders/sec \\ Latency &= 1\ msec \end{aligned} \tag{8}$$

(4) Comparing Performance

$$\begin{aligned} Speedup &= N \\ N &= \frac{Speed(x)}{Speed(y)} = \frac{Latency(y)}{Latency(x)} = \frac{Throughput(x)}{Throughput(y)} \end{aligned} \tag{9}$$

(5) Quiz: Performance Comparison 1

- Laptop takes 4 hours to compress video
- New laptop takes 10 minutes

$$Speedup = \frac{(4 \times 60)}{10} = 24 \tag{10}$$

(6) Quiz: Performance Comparison 2

The new laptop broke, so you have to go back to using the slower laptop.

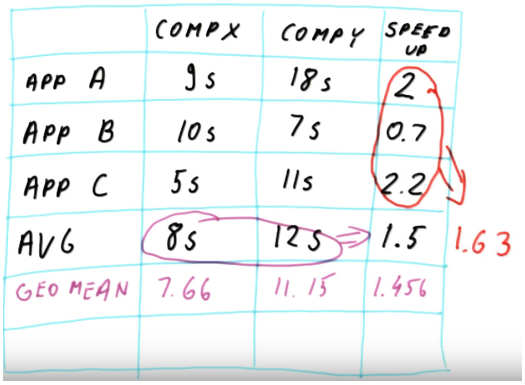
$$Speedup = \frac{10}{(4 \times 60)} = 0.042 \tag{11}$$

(7) Speedup

$$\begin{aligned} Speedup > 1 &\Rightarrow Improved\ Performance \\ Speedup < 1 &\Rightarrow Worse\ Performance \\ Performance &\sim Throughput \\ Performance &\sim \frac{1}{Latency} \end{aligned} \tag{12}$$

(13) Summarizing Performance

Use the *geometric mean* instead of the *arithmetic mean* for comparing speedups.



(14) Quiz: Speedup Averaging

- Speedup of 2
- Speedup of 8

$$Overall\ Speedup = \sqrt{2 \times 8} = 4 \tag{13}$$

(15) Iron Law of Performance

$$\begin{aligned} CPU\ Time = &\#\ of\ Instructions \\ &\times Cycles\ per\ Instruction \times Clock\ Cycle\ Time \end{aligned} \tag{14}$$

(16) Quiz: Iron Law 1

- 3 Billion Instructions
- 2 Cycles / Instruction
- Processor Clock is 3 GHz

$$Execution\ Time = 3\ billion \times 2 \times \frac{1}{3\ billion} = 2\ sec \tag{15}$$

(17) Iron Law for Unequal Instruction Times

$$CPU\ Time = \left(\sum_i = IC_i \times CPI_i \right) \times frequency \tag{16}$$

Quiz: Iron Law 2

- 10 billion branches (CPI = 4)
- 15 billion loads (CPI = 2) 5 billion stores (CPI = 3)
- 20 billion integer adds (CPI = 1)
- Clock at 4 GHz

$$Execution\ Time = \frac{10b \times 4 + 15b \times 2 + 5b \times 3 + 20b \times 1}{4b} = 26.25\ sec \tag{17}$$

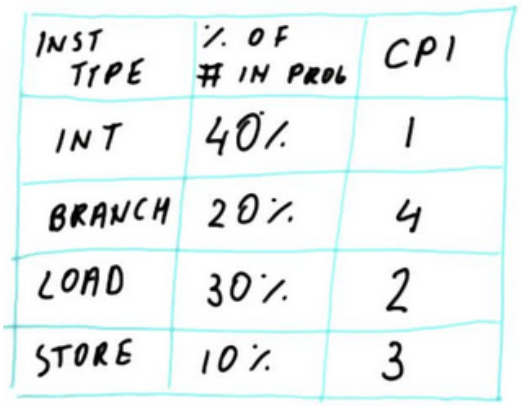
(19) Amdahl's Law

Calculates the overall speedup when a portion of a program is enhanced.

$$Frac_{ENH} = \%\ of\ original\ execution\ time\ that\ is\ affected\ by\ the\ enhancement$$
$$Speedup = \frac{1}{(1 - Frac_{ENH}) + \frac{Frac_{ENH}}{Speedup_{ENH}}} \tag{18}$$

(20) Quiz: Amdahl's Law

- 50 billion instructions
- 2 GHz
- Improve branch from 4 to 2 CPI



INST TYPE	% OF # IN PROG	CPI
INT	40%	1
BRANCH	20%	4
LOAD	30%	2
STORE	10%	3

$$Frac_{ENH} = \frac{0.2 \times 4}{0.4 \times 1 + 0.2 \times 4 + 0.3 \times 2 + 0.1 \times 3} \approx 0.381$$
$$Speedup = \frac{1}{(1 - 0.381) + \frac{0.381}{\frac{4}{2}}} \approx 1.24 \tag{19}$$

(22) Quiz: Amdahl's Law 2

INST TYPE	% OF TIME	CPI
INT	40%	1
BR	20%	4
LD	30%	2
ST	10%	3

(i) Branch CPI 4 → 3

$$Speedup = \frac{1}{0.8 + \frac{0.2}{4/3}} = 1.05 \tag{20}$$

(ii) Increase Clock Frequency 2 → 2.3 GHz

$$Speedup = \frac{1}{0 + \frac{1}{2.3/2}} = 1.15 \tag{21}$$

(iii) Store CPI 3 → 2

$$Speedup = \frac{1}{0.9 + \frac{0.1}{3/2}} = 1.034 \tag{22}$$

Lhamda's Law

- Amdahl: Make common case fast
- Lhamda: Do not mess up uncommon case too badly

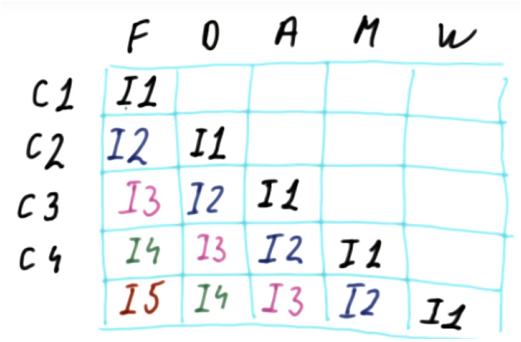
Example

$$Speedup = \frac{1}{\frac{0.1}{0.1} + \frac{0.9}{2}} = 0.7 \tag{23}$$

(27 - 37) Problem Set

Lesson 4: Pipelining

(3) Pipelining in a Processor



(4) Quiz: Laundry Pipelining

- Wash (1 hour) → Dry (1 hour) → Fold (1 hour)
- 10 loads of laundry

$$\begin{aligned} No\ Pipelining &\Rightarrow 3 \times 10\ hours = 30\ hours \\ With\ Pipelining &\Rightarrow 1 \times 3\ hours + 9 \times 1\ hour = 12\ hours \end{aligned} \tag{24}$$

(5) Instruction Pipelining

- 5-stage pipeline (1 cycle/stage)
- 10 instructions

$$\begin{aligned} No\ Pipelining &\Rightarrow 10 \times 5\ cycles = 50\ cycles \\ With\ Pipelining &\Rightarrow 1 \times 5\ cycles + 9 \times 1\ cycle = 14\ cycles \end{aligned} \tag{25}$$

(8) Processor Pipeline Stalls and Flushes

(10) Quiz: Control Dependencies

- 25% of all instructions are taken branches with jumps
- 10-stage pipeline
- Correct target for branch/jump computed in 6^{th} stage

$$Actual\ CPI = 0.75 + 0.25 \times 6 = 2.25 \tag{26}$$

subsect(11) Data Dependencies

- Read after write → True Dependency
- Write after write and write after read → False Dependency

(12) Quiz: Data Dependencies

I1: MUL R1,R2,R3
I2: ADD R4,R4,R1
I3: MUL R1,R5,R6
I4: SUB R4,R4,R1

	RAW	WAR	WAW
I1 → I2	✓	-	-
I1 → I3	-	-	✓
I1 → I4	-	-	-
I2 → I3	-	✓	-

(13) Dependencies and Hazards

- Dependencies - Instructions in a program that share the same registers.
- Hazards - Dependencies that can result in incorrect execution.

(14) Quiz: Dependencies and Hazards

ADD R1,R2,R3	Dependency?	Hazard?
SUB R5,R1,R4	✓	✓
DIV R6,R1,R7	✓	-
MUL R7,R1,R8	✓	-

(15) Handling Hazards

- Detect hazardous situations
  - Flush dependent instructions
  - Stall dependent instructions
  - Fix values read by dependent instructions

(16) Quiz: Flushes, Stalls, and Forwarding

$$Fetch \rightarrow Read \rightarrow ALU/BR \rightarrow Mem \rightarrow Wr \tag{27}$$

	BNE R1,R0,Label	Flush	Stall	Forward
	ADD R4,R5,R6 SUB R5,R4,R3	✓	-	-
Label:	MUL R1,R2,R3 LW R1,0(R1)	-	-	✓
	ADD R1,R1,R1	-	✓	✓

(17) How Many Stages?

- Ideal CPI = 9
- More stages means:
  - $More\ Hazards \Rightarrow CPI \uparrow$
  - $Less\ Work/Stage \Rightarrow CycleTime \downarrow$

$$\begin{aligned} Exe\ Time &= \#Inst \times Cycle\ Time \\ \#\ Stages &\Rightarrow Balance\ CPI\ \&\ Cycle\ Time \\ Performance &\Rightarrow 30-40\ stages \\ Performance + Power &\Rightarrow 10-15\ stages \end{aligned} \tag{28}$$

(20 - 30) Problem Set

Lesson 5: Branches

(3) Branch Prediction Requirements

- Know only PC of instruction
  - What is the PC of the next instruction to fetch
- Must Correctly Guess:
  - Is this a taken branch?
  - If taken, what is the target PC?

(4) Branch Prediction Accuracy

$$CPI = 1 + \frac{mispreds}{inst} \times \frac{penalty}{mispred} \tag{29}$$

- 20% of all instructions are branches

Accuracy ↓	Resolve BR in 3^{rd} Stage	Resolve BR in 10^{th} Stage
50% for BR 100% for other	$1 + 0.5 \times 0. \times 2$ 1.2	$1 + 0.5 + 0.2 \times 9$ 1.9
90% for BR 100% for all other	$1 + 0.1 \times 0.2 \times 2$ 1.04	$1 + 0.1 \times 0.2 \times 9$ 1.18
Speedup	1.15	1.61

(5) Quiz: Branch Prediction Benefit

- 5 stage pipeline
- Branch resolved in 3^{rd} stage
- Fetch nothing until sure what to frtch
- Execute many iterations of:

LOOP:	Cost No Pred	Perfect Pred
ADD R1,R1,-1	2	1
ADD R2,R2,R2	2	1
BNEZ R1,LOOP	3	1
Total	7	3

$$Speedup = \frac{7}{3} \approx 2.33 \tag{30}$$

(6) Performance with Not-taken Prediction

- Refuse to predict
  - Branch: 3 cycles
  - Non-branch: 2 cycles
- Predict not-taken:
  - Branch: 1 or 3 cycles
  - Non-branch: 1 cycle

Quiz: Multiple Predictions

$$Fetch \rightarrow Decode \rightarrow ALU\ (BR\ Resolved) \rightarrow Mem \rightarrow WR \tag{31}$$

- Using not-taken predictor

```
BNE R1,R2,LABEL A    (Taken)
BNE R1,R3,LABEL B    (Taken)
A
B
C
LABEL A:
X
Y
LABEL B:
Z
```

$$Cycles\ Wasted\ on\ Mispredictions = 2 \tag{32}$$

(8) Predict Not-Taken

- Increment PPC
- 20% of instructions are branches
- 60% of branches are taken
- Correct: 80% + 8%, Incorrect: 12%

$$CPI = 1 + 0.12 \times Penalty \tag{33}$$

(9) Why We Need Better Prediction

	Not Taken (88%)	Better (99%)	Speedup
5-stage pipe (3^{rd} stage)	$1 + 0.12 \times 2$ 1.24	$1 + 0.01 \times 2$ 1.02	1.22
14-stage (11^{th} stage)	$1 + 0.12 \times 10$ 2.2	$1 + 0.01 \times 10$ 1.1	2
11^{th} stage 4 inst/cycle	$0.25 + 0.12 \times 10$ 1.45	$0.25 + 0.01 \times 10$ 0.35	4.14

(10) Quiz: Predictor Impact

- Pentium 4 "Prescott":
  - FETCH, ... 29 stages ... , Resolve branches
  - Branch prediction
  - Multiple instruction/cycle
- Program:
  - 20% of instructions are branches
  - 1% if branches are mispredicted
  - CPI = 0.5

Perfect branch predictor:

$$CPI = 0.5 - 0.2 \times 0.01 \times 30 = 0.44 \tag{34}$$

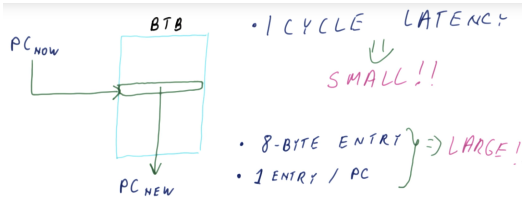
If 2% of branches are mispredicted:

$$CPI = 0.44 + 0.2 \times 0.02 \times 30 = 0.56 \tag{35}$$

(12) Better Prediction – How?

$$PC_{next} = f(PC_{now}, History[PC_{now}]) \tag{36}$$

(13) BTB – Branch Target Buffer



(14) Realistic BTB

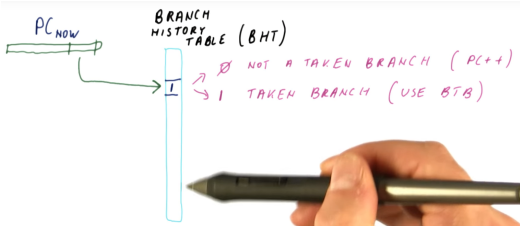
- Map least significant bits in PC_{now} to an index in the BTB

(15) Quiz: BTB

- BTB has 1024 entries
- Fixed-size 4-byte instructions, word-aligned
- 32-bit address
- $PC = 0x0000AB0C$

$$BTB\ Index = 10 least Sig Uniq Bits(PC) = 0b1011000011 = 0x2C3 \quad (37)$$

(16) Direction Predictor



(17 - 21) Quiz: BTB and BHT

- BHT: 16 entries, perfect prediction
- BTB: 4 entries, perfect prediction

			Times BHT Accessed	BHT Entry	Times BTB Accessed	BTB Entry Used	Missed Preds. if 1-bit BHT
0xC000		MOV R2,100					
0xC004		MOV R1,0	1	0	0		
0xC008	LOOP:	BEQ R1,R2,DONE	101	2	1	2	1
0xC00C		ADD R4,R3,R1	100	3	0		
0xC010		LW R4,0(R4)	100	4	0		
0xC014		ADD R5,R5,R4	100	5	0		
0xC018		ADDR1,R1,1	100	6	0		
0xC01C		MOV R1,0	100	7	100	3	1
0xC020	DONE:						

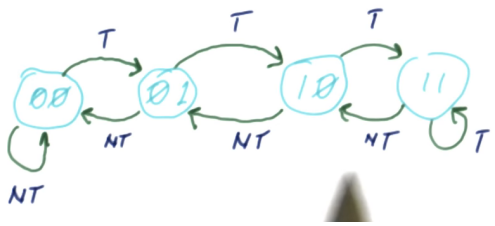
(22) Problems with 1-Bit Prediction

- Predicts well:
  - Always taken
  - Always not taken
  - $Taken >>> Not\ Taken$
  - $Taken <<< Not\ Taken$
- Not so well:
  - $Taken > Not\ Taken$
  - $Not\ Taken > Taken$
  - Short loop
- Pretty bad:
  - $Taken \approx Not\ Taken$

$$Each\ Anomoly \Rightarrow Two\ Mispredictions \tag{38}$$

(23) 2-Bit Predictor

- 00: Strong Not Taken
- 01: Weak Not Taken
- 10: Weak Taken
- 11: Strong Taken



(25) Quiz: 2BP

- 2-bit predictor
- Start in strong not-taken state
- A sequence always resulting in incorrect prediction:

T T N T N ...

(26) 1BP,2BP

- $1BP \rightarrow 2BP$ 3BP? 4BP?
  - Bad: Cost ↑
  - Good: When "anomalous" outcomes come in streaks
    - How often does this happen?
- Stay with 2BP, maybe 3BP

(28) 1-Bit History with 2-Bit Counters

State	Pred	Outcome	Correct?
(0, SN, SN)	N	T	×
(1, WN, SN)	N	N	✓
(0, WN, SN)	N	T	×
(1, WT, SN)	N	N	✓
(0, WT, SN)	T	T	✓
(1, ST, SN)	N	N	✓
(0, ST, SN)	T	T	✓

(29) Quiz: 1-Bit History

- 1-Bit history (Start 0)
- 2-BC/history (Start SN)
- (NNT)*
- Continues for 100 repitions

$$\#\ Mispredictions = 100 \tag{39}$$

(30 - 31) N-Bit History Predictors

- Works for all patterns of $length \leq N + 1$
- Costs $N + 2 \times 2^N$ per entry
- Most BCs are wasted

(32) Quiz: N-Bit History Predictor

- N-Bit history, 2BCs/History
- Need 1024 entries

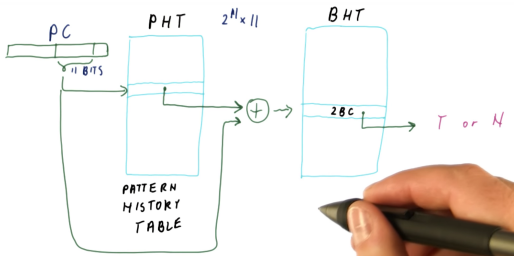
	cost(bits)	(NNNT)*	# of 2BCs used for (NT)*
N = 1	5×1024	bad	2
N = 4	$(4 + 2 \times 2^4) \times 1024$	good	2
N = 8	532,480	good	2
N = 16	134,234,112	good	2

(33) Quiz: History Predictor

```
for(int i = 0; i != 8; i++)
    for(int j = 0; j != 8; j++)
        doSomething();
```

- The history predictor needs at least 4 entries
- Each entry needs at least an 8-bit history (246 2-bit counters)

(34 - 36) History with Shared Counters



$$\begin{aligned} TTTT &\Rightarrow 1111 \Rightarrow 1\ counter \\ NNNN &\Rightarrow 0000 \Rightarrow 1\ counter \\ NTNT &\Rightarrow 010101\ or\ 101010 \Rightarrow 2\ counters \\ NNNNT &\Rightarrow 16 \times H \Rightarrow 16\ counters \end{aligned} \tag{40}$$

(37) PShare

PShare

- Private history
- Shared counters
- Good for:
  - Even-odd
  - 8-iteration loop

GShare

- Global history
- Shared counters
- Good for:
  - Correlated branches

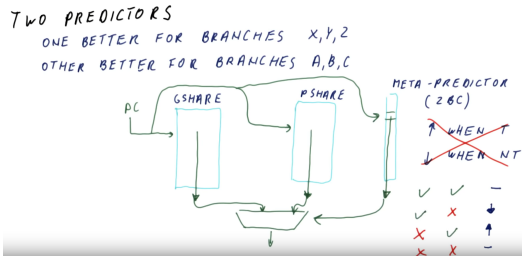
(38) Quiz: PShare vs. GShare

```
for(int i = 1000; i != 0; i–)
    if(i % 2)
        n += i;
           ⇓
LOOP:
BEQ R1,0,EXIT
AND R2,R1,1
BEQ R2,0,EVEN
ADD R3,R3,R1
ADD R1,R1,-1
B LOOP
EXIT:
```

$$\begin{aligned} History\ for\ PShare &= 1 \\ History\ for\ GShare &= 3 \end{aligned} \tag{41}$$

(40) Tournament Predictor

- Two Predictors
  - One better for branches x, y, z
  - Other better for branches a, b, c



(41) Hierarchical Predictor

- Tournament
  - 2 good predictors
  - update both on each dicision
- Hierarchical
  - 1 good, 1 ok predictor
  - Update ok-predictor on each decision
  - Update good-predictor onlu if the ok-predictor is not sufficient.

(43) Quiz: Multi-Predictor

- 2 BP works fine for 95% of instructions
- PShare works fine for the same 95%, plus 2% more
- GShare works fine for the same 95%, plus the other 3%

The overall predictor is a Hierarchical predictor that chooses between a 2BP and a tournament predictor, which itself chooses between a PShare and a GShare.

(45) RAS

- Contains the return addresses of functions
- When the RAS is full, there are two choices:
  - Don't push
  - Wrap around

(46) Quiz: RAS Full

When the RAS is full, it is better to wrap around than to not push.

(47) But How Do We Know it's a RET

- Predictor
  - Must be very accurate
  - PC = 0xABCD was RET $\Rightarrow$ 0xABCD is RET now
- Precode
  - Annotate instruction in cache

(49 - 53) Problem Set

Lesson 6: Predication

(2) Prediction

Branch Prediction

- Guess where it is going
- No penalty if correct (0% waste)
- Huge panalty if wrong (50% waste)

Predication

- Do work of both directions
- Wast up to 50%
- Throw away work from wrong path

Loop ⇒ Predict
Func ⇒ Predict
Large if-then-else ⇒ Predict
Small if-then-else ⇒ Predicate

(3) If Conversion

```
if(cond){
    x = arr[i];
    y = y + 1;
}else{
    x = arr[j];
    y = y - 1;
}
```

⇓

```
x1 = arr[i];
x2 = arr[j];
y1 = y + 1;
y2 = y - 1;
x = cond ? x1 : x2; ⇒ MOV x,x1,cond
y = cond ? y1 : y2; ⇒ MOV y,y1,cond
```

(4) Conditional Move

MIPS

- MOVZ $R_d, R_s, R_t \Rightarrow$ if($R_t == 0$) $R_d = R_s$;
- MOVN $R_d, R_s, R_t \Rightarrow$ if(R_t != 0) $R_d = R_s$;

x86

- CMOVZ, CMOVNZ, CMOVGT, ... $\Rightarrow$ if(cond) Dst = Src;

(5, 7) Quiz: MOVZ, MOVN, and Performance

BEQZ R1,ELSE (taken 50% of the time)
ADDI R2,R2,1
B END
ELSE:
ADDI R3,R3,1
END:
⇓
ADDI R4,R2,1
ADDI R5,R3,1
MOVN R2,R4,R1
MOVZ R3,R5,R1

- 30-instruction penalty

$$Performance\ Threshold = \frac{4 - (2+3) \times 0.5}{30} = 5\%\ Accuracy \tag{42}$$

If-conversion is better when prediction accuracy is below 95%.

(8) MOVc Summary

- Needs compiler support
- Removes hard-to-predict branches
- More registers are needed (unless full predication is possible)
  - Results from both paths
- More instructions are executed
  - Both paths
  - MOVc to select actual results (unless full predication is possible)

Full predication is achieved when all instructions are conditional. This requires extensive support in the instruction set.

(9) Full Predication HW Support

MOVcc

- Separate opcode

Full Predication

- Add condition bits to every instruction

(10) Full Predication Example

```
BEQZ R1,ELSE
ADDI R2,R2,1
B END
ELSE:
ADDI R3,R3,1
END:
        ⇓ (if-convert)
MP.EQZ P1,P2,R1
(P2) ADDI R2,R2,1
(P1) ADDI R3,R3,1
```

(11) Quiz: Full Predication

```
BEQZ R2,ELSE
ADD R1,R1,1
B DONE
ELSE:
ADD R1,R1,-1
DONE:
```

⇓ (if-convert)

```
MP.EQZ P1,P2,R2
(P2) ADD R1,R1,1
(P1) ADD R2,R2,1
```

- CPI = 0.5 without mispredictions
- Misprediction penalty of 10 cycles

$$\begin{aligned} Performance\ Threshold &= \frac{(3 \times 0.5) - (2+3) \times 0.5 \times 0.5}{10} \\ &= 2.5\%\ Accuracy \end{aligned} \tag{43}$$

Do if-conversion if BEQZ prediction is less than 97.5% correct.

(13 - 32) Problem Set

Lesson 7: ILP

(2) ILP All in the Same Cycle

Instruction	Cycle 1	2	3	...	N
R1 = R3 + R3	FETCH	DECODE	EXECUTE	...	WRITE
R4 = R1 - R5	FETCH	DECODE	EXECUTE	...	WRITE
R6 = R7 $\oplus$ R8	FETCH	DECODE	EXECUTE	...	WRITE
R5 = R8 $\times$ R9	FETCH	DECODE	EXECUTE	...	WRITE
R4 = R8 + R9	FETCH	DECODE	EXECUTE	...	WRITE

$$CPI = \frac{N\ cycles}{\infty\ instructions} = 0 \tag{44}$$

- Data dependencies make this impossible

(3) The Execute Stage

- I2 depends on I1

	EXE
I1	I1
I2	stall
I3	I3
I4	I4
I5	I5

$$CPI = \frac{2\ cycles}{5\ instructions} = 0.4 \tag{45}$$

(6) Quiz: Dependency

- 5-stages (FETCH, RR, EXE, MEM, WREG)
- Forwarding enabled
- 10 inputs in each stage

	EXE Cycle	WB
MUL R2,R2,R2	2	4
ADD R1,R1,R2	3	5
MUL R3,R3,R3	2	4
ADD R1,R1,R3	4	6
MUL R4,R4,R4	2	4
ADD R1,R1,R4	5	7

(7) Removing False Dependencies

- RAW – True Dependencies
- WAR, WAW – False (Name) Dependencies

(8) Duplicating Register Values

	c100	c101	c102	c103
R1 = R2 + R3	EXE	MEM	WR	-
R4 = R1 - R5	-	EXE	MEM	WR
R3 = R4 + 1	-	-	EXE	MEM
R4 = R8 - R9	EXE	MEM	WR	-
⋮	⋮	⋮	⋮	⋮
$\cdots$ = R4 $\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

- Create copies of shared registers

(9) Register Renaming

- *Architectural Registers*: Registers a programmer or compiler use
- *Physical Registers*: All places a value can go
- Rewrite a program to use physical registers
- *Register Allocation Table (RAT)*: A table that says which physical register has value for which architectural register

(10) RAT Example

Initial RAT

0	P0
1	P1
2	P2
3	P3
4	P4
5	P5
6	P6
⋮	...

Instructions

Fetched	Renamed
ADD R1,R2,R3	ADD P17,P2,P3
SUB R4,R1,R5	SUB P18,P17,P5
XOR R6,R7,R8	XOR P19,P7,P8
MUL R5,R8,R9	MUL P20,P8,P9
ADD R4,R8,R9	ADD P21,P8,P9

(11) Quiz: Register Renaming

Final RAT

R1	P8
R2	P11
R3	P10
R4	P4
R5	P12
R6	P6

Instructions

Fetched	Renamed
MUL R2,R2,R2	MUL P7,P2,P2
ADD R1,R1,R2	ADD P8,P1,P7
MUL R2,R4,R4	MUL P9,P4,P4
ADD R3,R3,R2	ADD P10,P3,P9
MUL R2,R6,R6	MUL P11,P6,P6
ADD R5,R5,R2	ADD P12,P5,P11

(13) Instruction Level Parallelism (ILP)

ILP = IPC When:

- Processor does entire instruction in 1 cycle
- Processor can do any number of instructions in the same cycle
- Has to obey true dependencies
- ILP is a property of a program, not a processor.

Steps to Get ILP:

1. Rename registors
2. "Execute"

(14) ILP Example

Instruction	Cycle
ADD P10,P2,P3	1
XOR P6,P7,P8	1
MUL P5,P8,P9	1
ADD P4,P8,P9	1
SUB P11,P10,P5	2

$$ILP = \frac{5\ instructions}{2\ cycles} = 2.5 \tag{46}$$

(15) Quiz: ILP

Instruction	Cylcle
ADD R1,R1,R1	1
ADD R2,R2,R1	2
ADD R3,R2,R1	3
ADD R6,R7,R8	1
ADD R8,R3,R7	4
ADD R1,R1,R1	2
ADD R1,R7,R7	1

$$ILP = \frac{7\ instructions}{4\ cycles} = 1.75 \tag{47}$$

(16) ILP With Structural & Control Dependencies

- No structural dependencies
- Perfect same-cycle branch prediction

(17) ILP vs. IPC

- ILP does not consider processor limitatipns, while IPC does
- ILP $\geq$ IPC

(18) Quiz: IPC & ILP

- 3-issue in-order
- 3 ALUs

Instruction	ILP Cylce	IPC Cycle
ADD R1,R2,R3	1	1
ADD R2,R3,R4	1	1
ADD R3,R1,R2	2	2
ADD R7,R8,R9	1	2
ADD R1,R7,R7	2	2
ADD R1,R4,R5	1	3

$$\begin{aligned} ILP &= \frac{6\ instructions}{2\ cycles} = 3 \\ IPC &= \frac{6\ instructions}{3\ cycles} = 2 \end{aligned} \tag{48}$$

(21 - 24) Problem Set

Lesson 8: Instruction Scheduling

(2) Improving IPC

- ILP can be good (>> 1)
- Control dependencies ⇒ branch prediction
- WAR & WAW data dependencies ⇒ register renaming
- RAW data dependencies ⇒ out of order execution
- Structural dependencies ⇒ invest in wider-issue

(3) Tomasulo's Algorithm

- Used in IBM 360
- Determine which instructions have inputs ready
- Register renaming
- Very similar to what we use today

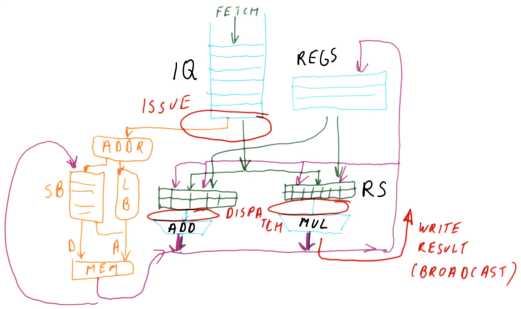
Tomasulo:

- Floating point instructions
- Fewer instructions in "window"
- No exception handling

Today:

- All instructions
- 100s of instructions
- Exception handling support

(4) The Picture



(5) Issue

- Take next (in program order) instruction from Issue Queue (IQ)
- Determine where inputs come from
- Get free reservation station (or correct kind)
- Put instruction into reservation station
- Tag destination register

(6) Issue Examples

IQ

4	F1 = F2 + F3
3	F4 = F1 - F2
2	F1 = F2 / F3
1	F2 = F4 + F1

RAT

F1	RS3
F2	FS1
F3	
F4	RS2

ADD

RS1	+	0.71	3.14
RS2	-	RS4	RS1
RS3	+	RS1	2.72

MUL

RS4	/	RS1	2.72
RS5			

RF

F1	3.14
F2	-1.00
F3	2.72
F4	RS2

(7) Quiz: Issue

IQ

2	F4 = F3 × F4
1	F4 = F1 / F2

RAT

F1	RS4
F2	FS1
F3	
F4	RS5

ADD

RS1	ADD	0.71	-1
RS2			
RS3			

MUL

RS4	DIV	RS1	2.72
RS5	DIV	RS4	RS1

RF

F1	3.14
F2	-1.00
F3	2.72
F4	0.71

(8) Dispatch

When a result is being broadcast:

1. Search for a matching tag
2. Broadcast result

(9) More than 1 Instruction Ready

- Oldest first
- Most dependencies first
- Random

(10) Quiz: Dispatch

- Dispatching RS1 = 0.29

ADD

RS1			
RS2	SUB	RS1	RS1
RS3	ADD	1.23	2.72

MUL

RS4	DIV	RS1	2.72
RS5			

RS3 wasn't dispatched already because it was either issued in the previous cycle or another instruction was dispatched in the ADD unit.

(11) Write Result (Broadcast)

1. Put Tag & Res on bus
2. Write to RF
3. Update RAT
4. Free RS

(12) More than 1 Broadcast

- Give priority to the slower operation

(13) Broadcast Stale Results

- Do not update the RAT or RF (Register File) with the result

(19) Quiz: Tomasulo's Algorithm

Tomasulo's Algorithm does not dispatch instructions or results in program order.

(20) Load & Store Instructions

- Dependencies Through Memory
  - RAW: SW to A, then LW from A
  - WAR: LW then SW
  - WAW: SW, then SW to same address
- What can we do?
  - Do loads and stores in order
  - Identify dependencies, reorder, etc.

(27) Timing Example

Load: 2 cycles
Add: 2 cycles
Multiply: 10 cycles
Divide: 40 cycles

	Instruction	IS	EX	WR	Comment
1	L.D F6,34(R2)	1	2	4	
2	L.D F2,45(R3)	2	4	6	
3	MUL.D F0,F2,F4	3	7	17	F2 from 2
4	SUB.D F8,F2,F6	4	7	9	F2 from 2
5	DIV.D F10,F0,F6	5	18	58	F0 from 3
6	ADD.D F6,F8,F2	6	10	12	F8 from 4

(28 - 29) Quiz: Tomasulo Timing

Latency:

- LD: 1 cycle
- ADD: 1 cylce
- MUL: 5 cycles

# of Reservation Stations:

- LD: 1
- ADD: 2
- MUL: 3

Same-cycle:

- Issue → Dispatch: No
- Capture → Dispatch: No
- RS freed → RS alloc: No

1	LD F6,0(R2)	1	2	3
2	MUL F2,F0,F1	2	3	8
3	ADD F6,F2,F6	3	9	10
4	ADD F6,F2,F6	4	11	12
5	ADD F1,F1,F1	11	12	13
6	ADD F1,F3,F4	13	14	15

(31 - 35) Problem Set

Lesson 9: ReOrder Buffer

(4) Correct Out of Order Execution

- Execute 000
- Broadcast 000
- But deposit values to REGs in-order!

⇒ ReOrder Buffer:

- Remembers program order
- Keeps result until it is safe to write to REGs

(5 - 6, 9) ROB

- Issue: Get Registration Station, get ROB entry, point RAT to ROB entry
- Dispatch: Ready? Send to EXE and free RS!
- BCast: Capture, WR to ROB
- Commit: WR to REG, update RAT

(7) Quiz: Free Reservation Station

If there is no ROB, it is more likely for an instruction to be unable to issue because there is no available RS.

(10) Quiz: ROB

The ROB is needed to:

- Remember the program order
- Temporarily store an instruction's result
- Serve as the name (TAG) for the result

(11) Branch Misprediction Recovery

LD R1,0(R1)
BNE R1,R2,Label
ADD R2,R1,R1
MUL R3,R3,R4
DIV R2,R3,R7

REGS

R1	700
R2	
R3	

RAT

~~ROB1~~	Issue
ROB5	Wrong
ROB4	Wrong

	Line	REG	Val	Done
Commit [1]	0			
Commit [2]	1	R1	700	✓
Commit [3], Issue [2]	2	-	ERROR	✓
	3	R2	✓	✓ (Wrong)
	4	R3	15	✓ (Wrong)
Issue [1]	5	R2	2	✓ (Wrong)

(13) ROB and Exceptions

- Treat an exception as a result
- Delay actual handling until the commit

(15) Quiz:Exceptions with ROB

Instruction	Status	New Status
ADD R2,R2,R1	committed	committed
LW R1,0(R2)	executing	committed
ADD R3,R4,R5	done	committed
DIV R3,R2,R3	executing (Exception)	unexecuted
ADD R1,R4,R4	done	unexecuted
ADD R3,R2,R2	done	unexecuted

(31) ROB Timing Example

- Free RS on BCast, not Dispatch
- Issue, Capture, Dispatch in same cycle
- ADD: 1 cycle, MUL: 10 cycles, DIV 40 cycles

Inst	Operands	IS	EXE	WR	Commit	Comment
DIV	R2,R3,R4	1	2	42	43	
MUL	R1,R5,R6	2	3	13	44	
ADD	R3,R7,R8	3	4	5	45	
MUL	R1,R1,R3	14	15	25	45	Need RS $\rightarrow$ Issue
SUB	R4,R1,R5	15	25	27	47	EXE depends on R1
ADD	R1,R4,R2	16	43	44	48	EXE depends on R2

(32 - 34) Quiz: ROB Timing

- BCast one ADD/SUB and one MUL/DIV per cycle
- Commit up to 2 instructions per cycle
- Free RS at dispatch
- ADD: 1 cycle, MUL: 2 cycles, DIV: 4 cycles

Inst	Operands	IS	EXE	WR	Commit
DIV	R2,R3,R4	1	2	6	7
MUL	R1,R5,R6	2	3	5	7
ADD	R3,R7,R8	3	4	5	8
MUL	R1,R1,R2	4	7	9	10
SUB	R4,R2,R5	5	7	8	10
ADD	R1,R4,R3	6	9	10	11

(36) Superscalar

- Fetch > 1 inst/cycle
- Decode > 1 inst/cycle
- Issue > 1 inst/cycle
- Dispatch > 1 inst/cycle
- Broadcast > 1 result/cycle
- Commit > 1 inst/cycle

(37) Terminology Confusion

- Issue ⇒ Allocate, Dispatch
- Dispatch ⇒ Execute, Issue
- Commit ⇒ Complete, Retire, Graduate

(38) Out of Order

Only execution and broadcast stages occur out-of-order.

(39) Quiz: In Order vs Out of Order

Stage	In-Order	Out-of-Order
Fetch	✓	
Decode	✓	
Issue	✓	
Dispatch		✓
Execute 1		✓
Execute 2		✓
Broadcast		✓
Commit	✓	
Release ROB Entry	✓	

Lesson 11: Memory Ordering

(2) Memory Access Ordering

- Eliminated control dependencies
  - Branch prediction
- Eliminated false dependencies on registers
  - Register renaming
- Obey raw register dependencies
  - Tomasulo-like scheduling
- What about memory dependencies?

(3) When Does Memory Write Happen?

- At commit
- Where does LOAD get data?
  - Load-Store Queue

(4 - 5) Load-Store Queue

Options:

1. In-order
2. Wait for all previous store addresses
3. Go anyway

(8) Quiz: Memory Ordering

	Out-of-Order		In-Order	
LW R1,0(R2)	1	41	1	41
SW R1,4(R2)		✓		✓
LW R1,0(R3)	2	42	43	83
SW R1,4(R3)		✓		✓
LW R1,0(R4)	3	43	86	125
SW R1,4(R4)		✓		✓

(9) Store To Load Forwarding

LOAD

- Which earlier store do we get values from?

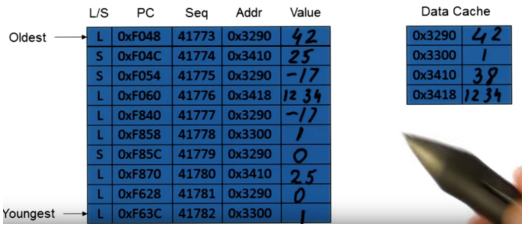
STORE

- Which later load do I give my value to?

Where do we figure this out?

- In the Load-Store Queue!

(10) LSQ Example



(11) LSQ, ROB, and RS

Issue LOAD/STORE:

- A ROB entry
- An LSQ entry

Issue non-LOAD/STORE:

- A ROB entry
- An RS

Execute LOAD/STORE

1. Compute address
2. Produce value

Write-result (LOAD)

- Broadcast

Commit LOAD/STORE

- Free ROB entries

+ Commit STORE

- Send write to memory

(12 -13) Quiz: Memory Ordering

SW R1 $\rightarrow$ 0(R2)
LW R2 $\leftarrow$ 0(R2)

LW does not access cache or memory because it shares the same addresses with the previous SW, so the values are forwarded from the Load-Store Queue.