Lecture: Review Session

- Topics: load-store queue wrap-up, first half recap

Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	4	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	8	3	abba		
LD R11 ← [R12]	1		abba		

Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	4	7	abba	5	commit
LD R7 ← [R8]	2		abce	3	6
ST R9 → [R10]	8	3	abba	9	commit
LD R11 ← [R12]	1		abba	2	10

Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	5	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	1	4	abba		
LD R11 ← [R12]	2		abba		

Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	5	7	abba	6	commit
LD R7 ← [R8]	2		abce	3	7
ST R9 → [R10]	1	4	abba	2	commit
LD R11 ← [R12]	2		abba	3	5

Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	4	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	8	3	abba		
LD R11 ← [R12]	1		abba		

Problem 4

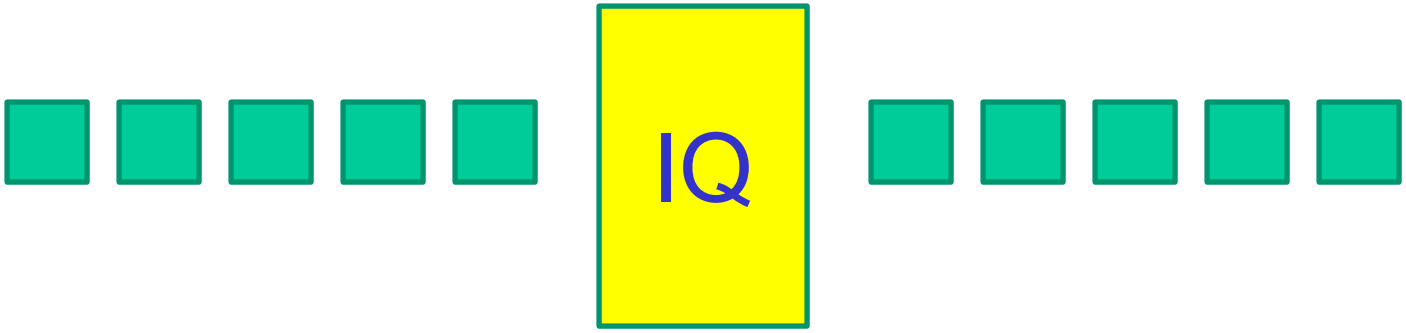
- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	4	7	abba	5	commit
LD R7 ← [R8]	2		abce	3	4
ST R9 → [R10]	8	3	abba	9	commit
LD R11 ← [R12]	1		abba	2	3/10

Solution:

LD/ST	The register for the address calculation is made available	The register that must be stored into memory is made available	The calculated effective addr	Addr calculation happens	Data memory accessed
LD	5	-	abcd	6	7
ST	7	4	abbb	8	at commit
LD	1	-	abbb	2	3/9
LD	3	-	abcd	4	5
ST	12	17	abbb	13	at commit
LD	2	-	abbb	3	4/9/18

OOO Example



Original code	Renamed code	InQ	Iss	Comp	Comm	Prev Map
ADD R1, R2, R3	ADD P33, P2, P3	i	i+1	i+6	i+6	P1
LD R2, 8(R1)	LD P34, 8(P33)	i	i+2	i+8	i+8	P2
ADD R2, R2, 8	ADD P35, P34, 8	i	i+4	i+9	i+9	P34
ST R1, (R3)	ST P33, (P3)	i	i+2	i+8	i+9	
SUB R1, R1, R5	SUB P36, P33, P5	i+1	i+2	i+7	i+9	P33
LD R1, 8(R2)	LD P1, 8(P35)	i+7	i+8	i+14	i+14	P36
ADD R1, R1, R2	ADD P2, P1, P35	i+9	i+10	i+15	i+15	P1

Problem 3

- Processor-A at 3 GHz consumes 80 W of dynamic power and 20 W of static power. It completes a program in 20 seconds.
  What is the energy consumption if I scale frequency down by 20%?

  What is the energy consumption if I scale frequency and voltage down by 20%?

Problem 3

- Processor-A at 3 GHz consumes 80 W of dynamic power and 20 W of static power. It completes a program in 20 seconds.
  What is the energy consumption if I scale frequency down by 20%?
  New dynamic power = 64W; New static power = 20W
  New execution time = 25 secs (assuming CPU-bound)
  Energy = 84 W x 25 secs = 2100 Joules

  What is the energy consumption if I scale frequency and voltage down by 20%?
  New dynamic power = 41W; New static power = 16W;
  New exec time = 25 secs; Energy = 1425 Joules

Problem 4

- Consider 3 programs from a benchmark set. Assume that system-A is the reference machine. How does the performance of system-B compare against that of system-C (for all 3 metrics)?

	P1	P2	P3
Sys-A	5	10	20
Sys-B	6	8	18
Sys-C	7	9	14

  - Sum of execution times (AM)
  - Sum of weighted execution times (AM)
  - Geometric mean of execution times (GM)

Problem 4

- Consider 3 programs from a benchmark set. Assume that system-A is the reference machine. How does the performance of system-B compare against that of system-C (for all 3 metrics)?

	P1	P2	P3	S.E.T	S.W.E.T	GM
Sys-A	5	10	20	35	3	10
Sys-B	6	8	18	32	2.9	9.5
Sys-C	7	9	14	30	3	9.6

  - Relative to C, B provides a speedup of 1.03 (S.W.E.T) or 1.01 (GM) or 0.94 (S.E.T)
  - Relative to C, B reduces execution time by 3.3% (S.W.E.T) or 1% (GM) or -6.7% (S.E.T)

Problem 6

- My new laptop has a clock speed that is 30% higher than the old laptop. I'm running the same binaries on both machines. Their IPCs are listed below. I run the binaries such that each binary gets an equal share of CPU time. What speedup is my new laptop providing?

	P1	P2	P3
Old-IPC	1.2	1.6	2.0
New-IPC	1.6	1.6	1.6

Problem 6

- My new laptop has a clock speed that is 30% higher than the old laptop. I'm running the same binaries on both machines. Their IPCs are listed below. I run the binaries such that each binary gets an equal share of CPU time. What speedup is my new laptop providing?

	P1	P2	P3	AM	GM
Old-IPC	1.2	1.6	2.0	1.6	1.57
New-IPC	1.6	1.6	1.6	1.6	1.6

AM of IPCs is the right measure. Could have also used GM. Speedup with AM would be 1.3.

Problem 2

- An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 sequential pipeline stages. The stages have the following lengths: 1ns; 0.6ns; 1.2ns; 1.4ns; 0.8ns. Answer the following, assuming that there are no stalls in the pipeline.

- What is the cycle time in the new processor?
- What is the clock speed?
- What is the IPC?
- How long does it take to finish one instr?
- What is the speedup from pipelining?
- What is the max speedup from pipelining?

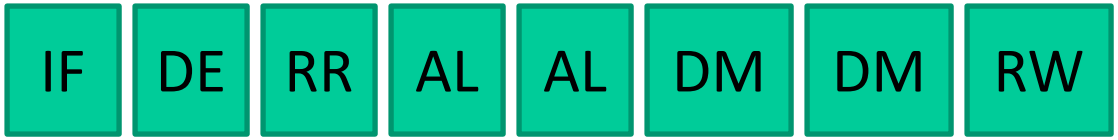
Problem 2

- An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 sequential pipeline stages. The stages have the following lengths: 1ns; 0.6ns; 1.2ns; 1.4ns; 0.8ns. Answer the following, assuming that there are no stalls in the pipeline.

- What is the cycle time in the new processor? 1.6ns
- What is the clock speed? 625 MHz
- What is the IPC? 1
- How long does it take to finish one instr? 8ns
- What is the speedup from pipelining? 625/192 = 3.26
- What is the max speedup from pipelining? 5.2/0.2 = 26

Problem 8

- Consider this 8-stage pipeline (RR and RW take a full cycle)

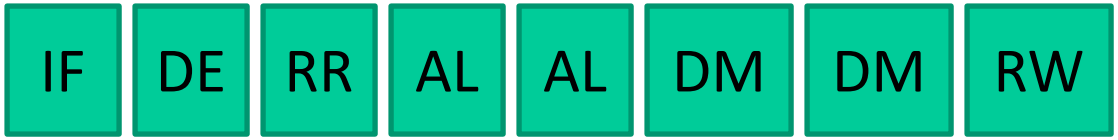


- For the following pairs of instructions, how many stalls will the 2nd instruction experience (with and without bypassing)?

  - ADD R3 ← R1+R2
    ADD R5 ← R3+R4
  - LD R2 ← [R1]
    ADD R4 ← R2+R3
  - LD R2 ← [R1]
    SD R3 → [R2]
  - LD R2 ← [R1]
    SD R2 → [R3]

Problem 8

- Consider this 8-stage pipeline (RR and RW take a full cycle)



- For the following pairs of instructions, how many stalls will the 2nd instruction experience (with and without bypassing)?

  - ADD R3 ← R1+R2
    ADD R5 ← R3+R4 without: 5 with: 1
  - LD R2 ← [R1]
    ADD R4 ← R2+R3 without: 5 with: 3
  - LD R2 ← [R1]
    SD R3 → [R2] without: 5 with: 3
  - LD R2 ← [R1]
    SD R2 → [R3] without: 5 with: 1

Problem 1

- Consider a branch that is taken 80% of the time. On average, how many stalls are introduced for this branch for each approach below:
  - Stall fetch until branch outcome is known
  - Assume not-taken and squash if the branch is taken
  - Assume a branch delay slot
    - You can't find anything to put in the delay slot
    - An instr before the branch is put in the delay slot
    - An instr from the taken side is put in the delay slot
    - An instr from the not-taken side is put in the slot

Problem 1

- Consider a branch that is taken 80% of the time. On average, how many stalls are introduced for this branch for each approach below:
  - Stall fetch until branch outcome is known – 1
  - Assume not-taken and squash if the branch is taken – 0.8
  - Assume a branch delay slot
    - You can't find anything to put in the delay slot – 1
    - An instr before the branch is put in the delay slot – 0
    - An instr from the taken side is put in the slot – 0.2
    - An instr from the not-taken side is put in the slot – 0.8

Problem 2

- Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 20-stage and 40-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.

Problem 2

- Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 1-stage, 20-stage and 50-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.

  - 1-stage: 1 instr every 5.1ns
  - 20-stage: first instr takes 0.35ns, the second takes 2.8ns
  - 50-stage: first instr takes 0.2ns, the second takes 4ns
  - Throughputs: 0.20 BIPS, 0.63 BIPS, and 0.48 BIPS

Problem 1

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

```
for (i=1000; i>0; i--)
   x[i] = y[i] * s;
```

Source code

```
Loop:   L.D       F0, 0(R1)        ; F0 = array element
        MUL.D     F4, F0, F2       ; multiply scalar
        S.D       F4, 0(R2)        ; store result
        DADDUI    R1, R1,# -8      ; decrement address pointer
        DADDUI    R2, R2,#-8       ; decrement address pointer
        BNE       R1, R3, Loop     ; branch if R1 != R3
        NOP
```

Assembly code

- How many cycles do the default and optimized schedules take?

Problem 1

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

```
for (i=1000; i>0; i--)
   x[i] = y[i] * s;
```

Source code

```
Loop:   L.D       F0, 0(R1)       ; F0 = array element
        MUL.D     F4, F0, F2      ; multiply scalar
        S.D       F4, 0(R2)       ; store result
        DADDUI    R1, R1,# -8     ; decrement address pointer
        DADDUI    R2, R2,#-8      ; decrement address pointer
        BNE       R1, R3, Loop    ; branch if R1 != R3
        NOP
```

Assembly code

Unoptimized: LD 1s MUL 4s SD DA DA BNE 1s -- 12 cycles
Optimized: LD DA MUL DA 2s BNE SD -- 8 cycles

Degree 2: LD LD MUL MUL DA DA 1s SD BNE SD
Degree 3: LD LD LD MUL MUL MUL DA DA SD SD BNE SD
– 12 cyc/3 iterations

Source Code:

```
for (i=1000; i>0; i--) {
w[i] = x[i] * w[i];
}
```

Assembly Code:

```
Loop:
L.D F1, 0(R1) // Get w[i]
L.D F2, 0(R2) // Get x[i]
MUL.D F1, F2, F1 // Multiply two numbers
S.D F1, 0(R1) // Store the result into w[i]
DADDUI R1, R1, #-8 // Decrement R1
DADDUI R2, R2, #-8 // Decrement R2
BNE R1, R3, Loop // Check if we've reached the end of the loop
NOP
```

Problem 3

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

```
for (i=1000; i>0; i--)
    x[i] = y[i] * s;
```

Source code

```
Loop:   L.D       F0, 0(R1)       ; F0 = array element
        MUL.D     F4, F0, F2      ; multiply scalar
        S.D       F4, 0(R2)       ; store result
        DADDUI    R1, R1,# -8     ; decrement address pointer
        DADDUI    R2, R2,#-8      ; decrement address pointer
        BNE       R1, R3, Loop    ; branch if R1 != R3
        NOP
```

Assembly code

- How many unrolls does it take to avoid stalls in the superscalar pipeline?

Problem 3

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

```
for (i=1000; i>0; i--)
   x[i] = y[i] * s;
```

Source code

```
Loop:   L.D      F0, 0(R1)      ; F0 = array element
        MUL.D    F4, F0, F2     ; multiply scalar
        S.D      F4, 0(R2)      ; store result
        DADDUI   R1, R1,# -8    ; decrement address pointer
        DADDUI   R2, R2,#-8     ; decrement address pointer
        BNE      R1, R3, Loop   ; branch if R1 != R3
        NOP
```

Assembly code

- How many unrolls does it take to avoid stalls in the superscalar pipeline?

```
LD
LD
LD      MUL
LD      MUL
LD      MUL
LD      MUL
LD      MUL
SD      MUL
```

7 unrolls. Could also make do with 5 if we moved up the DADDUIs.

Problem 2

- What is the storage requirement for a tournament predictor that uses the following structures:
  - a “selector” that has 4K entries and 2-bit counters
  - a “global” predictor that XORs 14 bits of branch PC with 14 bits of global history and uses 3-bit counters
  - a “local” predictor that uses an 8-bit index into L1, and produces a 12-bit index into L2 by XOR-ing branch PC and local history. The L2 uses 2-bit counters.

Problem 2

- What is the storage requirement for a tournament predictor that uses the following structures:
  - a “selector” that has 4K entries and 2-bit counters
  - a “global” predictor that XORs 14 bits of branch PC with 14 bits of global history and uses 3-bit counters
  - a “local” predictor that uses an 8-bit index into L1, and produces a 12-bit index into L2 by XOR-ing branch PC and local history. The L2 uses 2-bit counters.

Selector = 4K * 2b = 8 Kb
Global = 3b * 2^14 = 48 Kb
Local = (12b * 2^8) + (2b * 2^12) = 3 Kb + 8 Kb = 11 Kb
Total = 67 Kb

Problem 3

- For the code snippet below, estimate the steady-state bpred accuracies for the default PC+4 prediction, the 1-bit bimodal, 2-bit bimodal, global, and local predictors. Assume that the global/local preds use 5-bit histories.

```
do {
    for (i=0; i<4; i++) {
      increment something
    }
    for (j=0; j<8; j++) {
      increment something
    }
    k++;
} while (k < some large number)
```

Problem 3

- For the code snippet below, estimate the steady-state bpred accuracies for the default PC+4 prediction, the 1-bit bimodal, 2-bit bimodal, global, and local predictors. Assume that the global/local preds use 5-bit histories.

```
do {
    for (i=0; i<4; i++) {
      increment something
    }
    for (j=0; j<8; j++) {
      increment something
    }
    k++;
} while (k < some large number)
```

PC+4: 2/13 = 15%
1b Bim: (2+6+1)/(4+8+1)
= 9/13 = 69%
2b Bim: (3+7+1)/13
= 11/13 = 85%
Global: (4+7+1)/13
= 12/13 = 92%
Local: (4+7+1)/13
= 12/13 = 92%