18-600 Recitation #6

Arch Lab (Y86-64 & O3 recap)

October 3rd, 2017

Arch Lab Intro

- Last week:
  - Part A: write and simulate Y86-64 programs
- This week:
  - Part B: optimize a Y86 program
  - Recap of O3
  - Intuition on how to work and analyze Part C i.e. parameter tuning

Part B

- You are supposed to modify only ncopy.ys
- Goals for **ncopy.ys**:
  - improve the performance by reducing the CPE (cycles per element, the number of cycles used / the number of elements copied)
  - Remember, each instruction in Y86 takes one cycle to execute

Optimizing a Y86 Program

- Reduce the CPE of ncopy by reducing the number of operations from inefficient code, data dependencies and conditional control flow
- Methods:
  - Streamline Code
  - Reduce Data Dependencies
  - Loop Unrolling
  - Jump Tables (and other control flow changes)

Streamline Code

- Reduce memory accesses

```
.L1
mrmovq (%rbx), %rax
...
jne .L1
```



```
mrmovq (%rbx), %rax
.L1
...
jne .L1
```

- Reducing number of register operations

```
.L1
irmovq $8, %rax
...
jne .L1
```



```
irmovq $8, %rax
.L1
...
jne .L1
```

Reduce Data Dependencies

- Results from one operation depend on the results of the other, stalling the pipeline execution
- Avoid clumping dependent operations together

```
irmovq $8, %r8
addq %r8, %r10
irmovq $10,
%r9  addq %r9,
%rbx
```



```
irmovq $8, %r8
irmovq $10, %r9
addq %r8, %r10
addq %r9, %rbx
```

Loop Unrolling

- Program transformation which **reduces the number of iterations** for a loop by increasing the number of elements computed on each iteration
- **Reduces the number of operations** that do not contribute directly to loop result, including loop indexing and conditional branching
- Exposes additional ways to optimize computation (multiple computations may be combined)

Recall: Loop Unrolling in C

```
void combine(vec_ptr v, int *dest)
{
      int sum = 0;
      int *data = get_vec_start(v);
      int length = vec_length(v);
      int i;
      for (i=0; i < length; i++) {
                sum += data[i];
      }
      *dest = sum;
}
```

Sum all integers without loop unrolling

```
void combine(vec_ptr v, int *dest)
{
      int sum = 0;
      int *data = get_vec_start(v);
      int length = vec_length(v);
      int limit = length - 1;
      int i;
      for (i=0; i < limit; i+=2) {
                sum += data[i] + data[i+1];
      }
      for (; i<length; i++)
          sum += data[i];
      *dest = sum;
}
```

Sum all integers with loop unrolling (Unroll by 2)

Loop Unrolling in example Y86 program

```
# long sum(long *start, long count)
# start in %rdi, count in %rsi

sum: irmovq $8, %r8    # Constant 8
     irmovq $1, %r9    # Constant 1
     xorq %rax,%rax     # sum = 0
     andq %rsi, %rsi    # Set CC
     jmp test          # Go To Test
Loop: mrmovq (%rdi),%r10 # Get *start
    addq %r10, %rax   # Add to Sum
    addq %r8, %rdi    # start++
    subq %r9, %rsi    # count--. Set
CC
test: jne loop         # Stop when 0
      ret
```

Loop Structure

Loop Unrolling (Unrolled by 2)

```
# long sum(long *start, long count)
# start in %rdi, count in %rsi
sum: irmovq $8, %r8       # Constant 8
     irmovq $2, %r9       # Constant 2
     xorq %rax,%rax       # sum = 0
     subq %r9, %rsi       # Set CC
     jmp test             # Go To Test
Loop: mrmovq (%rdi),%r10 # Get *start
     addq %r10, %rax      # Add to Sum
     addq %r8, %rdi       # start++
     mrmovq (%rdi),%r10
     addq %r10, %rax      # Add to Sum
     addq %r8, %rdi       # start++
     subq %r9, %rsi       # count-=2. Set CC
test: jge Loop            # Stop when < 0
      addq %r9, %rsi      #
      je done             # Goto done if 0
      mrmovq (%rdi),%r10 # Get *start
      addq %r10, %rax     # Add to Sum
done: ret
```

For count >= 2

Loop Structure unrolled by 2

Handle overflow cases
(when count % 2 != 0)

Jump Tables

- Jump table is an abstract data structure to transfer control
- It is an array where each entry is the address of a code segment
- Each code segment implements set of actions which should be taken when a particular condition is satisfied
- Time taken is invariant of the number of switch cases

Jump Tables: Example

```
void switch_eg(long x, long n, long *dest)
{
        long val = x;

        switch (n) {

                    case 100:
                     val *= 13;
                      break;

                    case 102:
                     val += 10;

                    case 103:
                     val += 11;
                      break;

                    case 104:
                    case 106:
                     val *= val;
                      break;

                     default:
                     val = 0;
                        }
        *dest = val;
}
```

Example switch statement

```
1       void switch_eg_impl(long x, long n, long *dest)
2       {
3           static void *jt[7] = {
4               &&loc_A, &&loc_def, &&loc_B,
5               &&loc_C, &&loc_D, &&loc_def,
6               &&loc_D
7           };
8           unsigned long index = n - 100;
9           long val;
10          if (index > 6)
11                goto loc_def;
12          goto *jt[index];
13      loc_A:          /* Case 100 */
14          val = x * 13;
15          goto done;
16      loc_B:          /* Case 102 */
17          x = x + 10;
18      loc_C:          /* Case 103 */
19          val = x + 1;;
20          goto done;
21      loc_D:          /* Case 104, 106 */
22          val = x * x;
23          goto done;
24      loc_def: /* Default case */
25          val = 0;
26      done:
27          *dest = val;
28      }
```

Translation into extended C
(labels as values)

Jump Tables: Example for X86

```
void switch_eg(long x, long n, long *dest)
{
        long val = x;

        switch (n) {

                case 100:
                    val *= 13;
                    break;

                case 102:
                    val += 10;

                case 103:
                    val += 11;
                    break;

                case 104:
                case 106:
                    val *= val;
                    break;

                default:
                    val = 0;
                }
        *dest = val;
}
```

Example switch statement

```
1         .section        .rodata
2         .align 8
3       .L4
4         quad .L3 Case 100
5         quad .L8 Case 101: default
6         quad .L5 Case 102
7         quad .L6 Case 103
8         quad .L7 Case 104
9         quad .L8 Case 105: default
10        quad .L7 Case 106
```

Jump Table

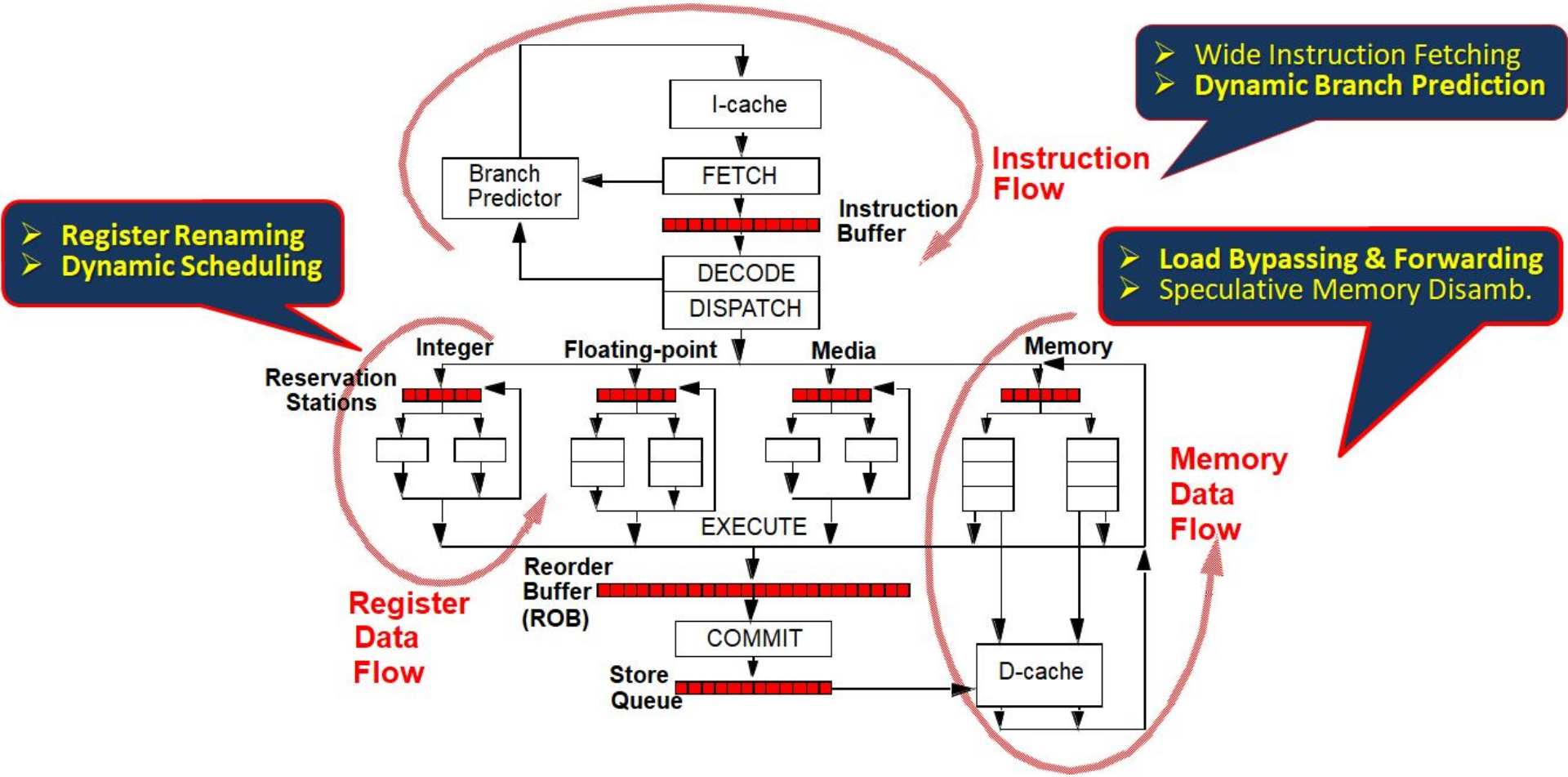
```
1       switch_eg:
2         subq $100, %rsi           Compute index = n - 100
3         cmpq $6, %rsi             Compare index:6
4         ja   .L8                  If >, goto loc_def
5         jmp  *.L4(,%rsi,8)        Goto *jg[index]
6       .L3:                        loc_A:
7         leaq (%rdi, %rdi, 2), %rax  3*x
8         leaq (%rdi, %rax, 4), %rdi  val = 13*x
9         jmp  .L2                    goto done
10      .L5:                        loc_B:
11        addq $10, %rdi              x = x + 10
12      .L6:                        loc_C:
13        addq $11, %rdi              val = x + 11
14        jmp .L2                     goto done
15      .L7                         loc_D:
16        imulq %rdi, %rdi            val = x * x
17        jmp .L2                     goto done
18      .L8:                        loc_def:
19        movl $0, %edi               val = 0
20      .L2:                        done:
21        movl $rdi, (%rdx)           *dest = val
22        ret                         Return
```

Assembly Code

NOTE: The assembly code and jumps with . prefixed labels will NOT work in Y86 since this is a primitive assembly language. Instead try to think along the lines of memory array and stack manipulation in Y86 to make a jump table.

O3 Recap

- ➔ Brief review of lecture material
- ➔ MIPS R10K Example (optional)

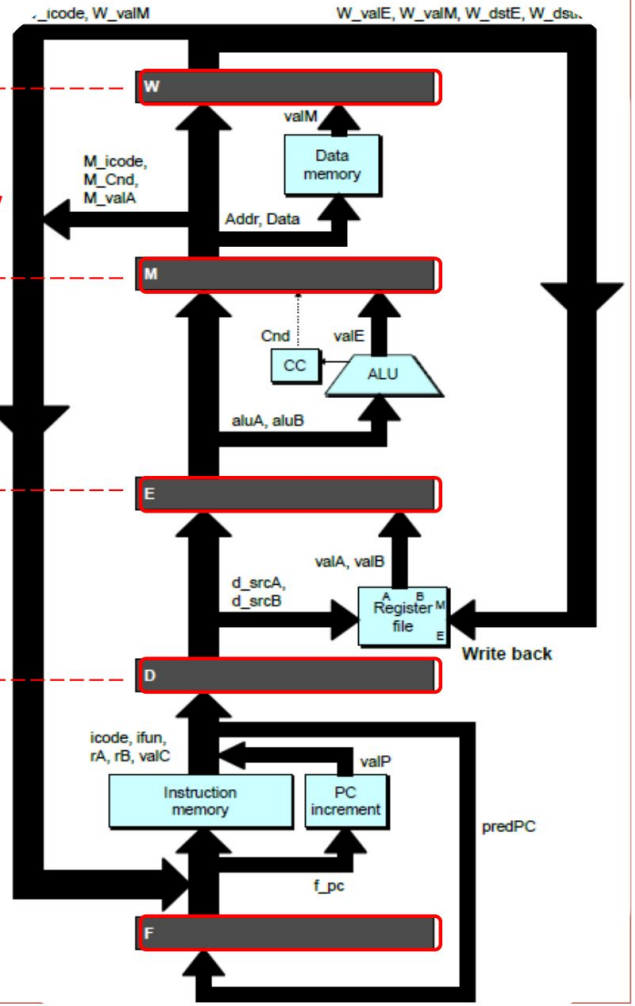

Wide Instruction Fetching
Dynamic Branch Prediction
Instruction Flow
I-cache
Branch Predictor
FETCH
Instruction Buffer
DECODE
DISPATCH
Register Renaming
Dynamic Scheduling
Load Bypassing & Forwarding
Speculative Memory Disamb.
Integer
Floating-point
Media
Memory
Reservation Stations
EXECUTE
Memory Data Flow
Reorder Buffer (ROB)
Register Data Flow
COMMIT
Store Queue
D-cache

Recap - Branch Prediction

- Problem:
  - Branch instructions gets “resolved” only after Execute stage. Assuming no branch prediction, subsequent instructions in the pipeline have to be abandoned if the condition is taken.
- Solution:
  - Speculates outcome of conditional instructions.
    - E.g. “jle”, “je”, “cmove” instructions in Y86-64.
  - If correctly predicted, we can avoid restarting pipeline and wasting processor cycles.

Dealing With Branches

			(i) **cond: PC←Y**	**5. Write back**
		(i) **cond: PC←Y**	~~(i+1) ← R1+R2~~	**4. Memory**
	(i) **cond: PC←Y**	(i+1) ←R1+R2	~~(i+2) ← R3+R4~~	**3. Execute**
(i) **cond: PC← Y**	(i+1) ← R1+R2	(i+2) ←R3+R4	~~(i+3) ← R5+R6~~	**2. Decode**
(i+1) ←R1+R2	(i+2) ← R3+R4	(i+3) ←R5+R6	(k) (target of br) fetch from M[Y]	**1. Fetch & PC update**



Recap - Branch Prediction

- Two prediction questions
  - Direction? - taken or not-taken
  - Target? - next PC to fetch
- Implemented with BTB (Branch Target Buffer)
  - Input: current PC + some history of past branches
  - Output: next PC
- Detailed explanation in lecture slides.

Recap - Register Renaming

- Anti (WAR) and output (WAW) dependencies are false dependencies. They can be eliminated if we have unlimited # of registers.
  - We do not actually have unlimited # of registers, but this is still possible because processors usually have much more *physical* registers than *architectural* registers.
  - E.g. the “Alpha” ISA has 32 integer and 32 floating-point *architectural* registers but the “Alpha 21264” microprocessor which implements this ISA, has 80 integer and 72 floating-point *physical* registers.

https://en.wikipedia.org/wiki/Alpha_21264

Recap - Register Renaming (example)

- Architectural registers: r1, r2, r3
- Physical registers: p1, p2, p3, p4, p5, p6, p7
- Identify all WAW's and WAR's, notice how renaming eliminates them

MapTable

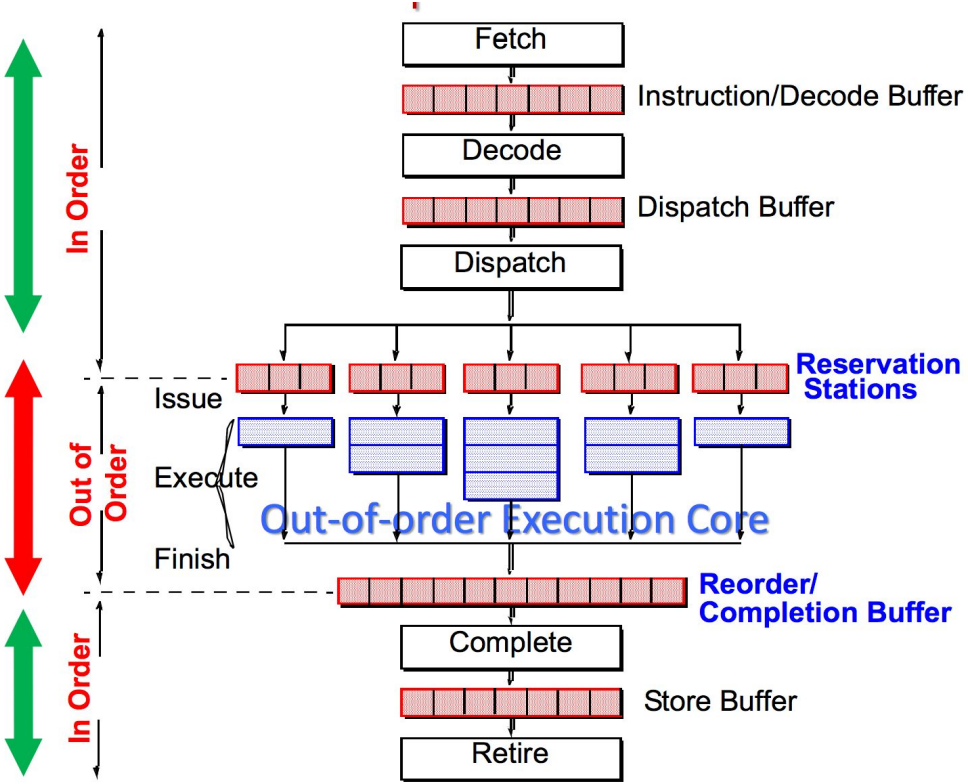
r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p6	p2	p5
p6	p7	p5

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

Raw insns	Renamed insns
add r2,r3→r1	add p2,p3→p4
sub r2,r1→r3	sub p2,p4→p5
mul r2,r3→r1	mul p2,p5→p6
div r1,r3→r2	div p6,p5→p7

Recap - Dynamic Scheduling



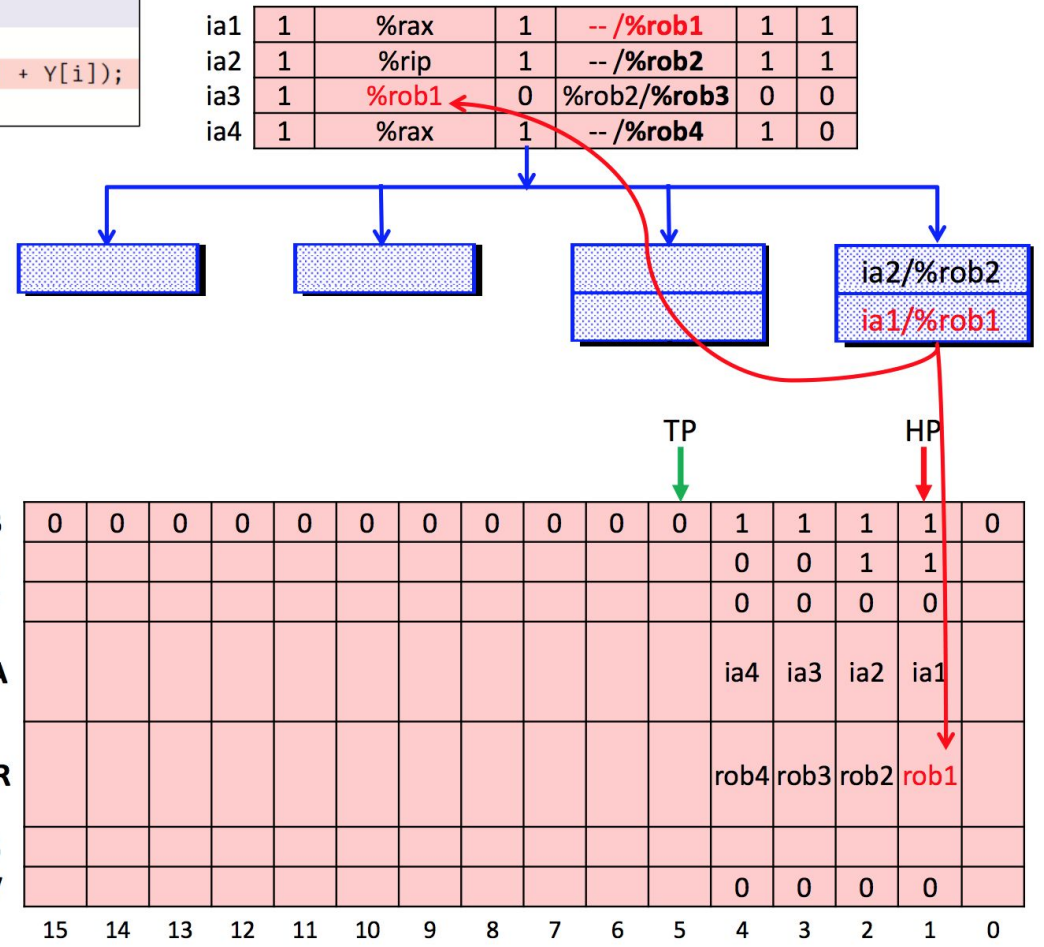
- Allow instructions to enter Execute stage out-of-order
- Program correctness (e.g. RAW dependencies) is ensured by Reservation Stations and Reorder Buffer.
- Understanding how they work is crucial for Arch Lab, Part C.

Cycle: 001

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

Four instructions (ia1, ia2, ia3, ia4) have been dispatched to RS with four corresponding entries allocated in ROB. ia1 and ia2 have been issued into LS FU. Next cycle, ia1 will finish and broadcast its result using tag “%rob1” to ia3.



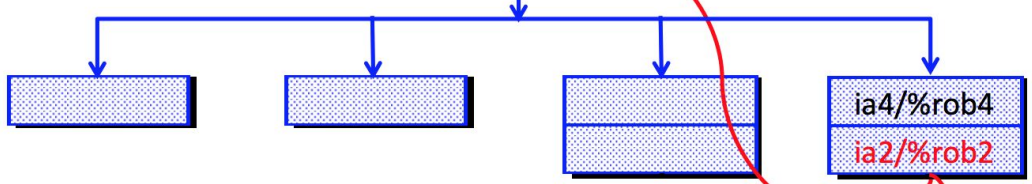
Cycle: 002

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

ia1 completes and is next to leave ROB.
Its RS entry has been reallocated to ia5.
ia4 is issued into LS FU.
Next cycle, ia2 will broadcast its results using tag "%rob2" to ia3.

ia5	1	%rob3	0	%rob4/**%rob5**	1	1
ia2	1	%rip	1	-- /**%rob2**	1	1
ia3	1	%rob1	1	%rob2/**%rob3**	0	0
ia4	1	%rax	1	-- /**%rob4**	1	1



TP HP

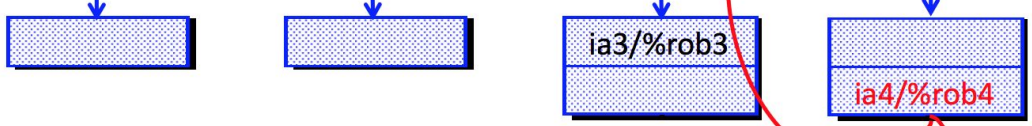
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
I											0	1	0	1	1	
F											0	0	0	0	1	
IA											ia5	ia4	ia3	ia2	ia1	
RR											Rob5	rob4	rob3	rob2	rob1	
S																
V											0	0	0	0	1	

Cycle: 003

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

ia5	1	%rob3	0	%rob4/%rob5	0	0
ia6	1	%rob5	0	--	1	0
ia3	1	%rob1	1	%rob2/%rob3	1	1
ia4	1	%rax	1	-- /%rob4	1	0



TP HP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
I										0	0	1	1	1		
F										0	0	0	0	1		
IA										ia6	ia5	ia4	ia3	ia2		
RR											rob5	rob4	rob3	rob2		
S																
V										0	0	0	0	1		

ia2 completes and is ready to leave ROB.
ia3 is now ready and is issued to MULT FU.
Next cycle, ia4 will forward result to ia5 in RS using tag “%rob4”.

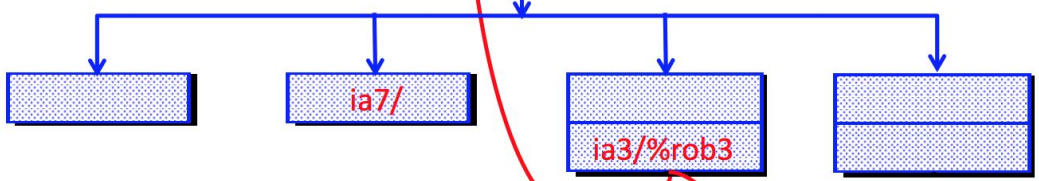
Cycle: 004

```
// Implment DAXPY here
for(i = 0; i < N; i++)
{
    Y[i] = (alpha * X[i] + Y[i]);
}
```

```
        movl $0, -4(%rbp)
        jmp .L2
.L3:
ia1     movsd X(,%rax,8), %xmm1
ia2     movsd alpha(%rip), %xmm0
ia3     mulsd %xmm1, %xmm0
ia4     movsd Y(,%rax,8), %xmm1
ia5     addsd %xmm1, %xmm0
ia6     movsd %xmm0, Y(,%rax,8)
ia7     addl $1, -4(%rbp)
.L2:
ia8     cmpl $2055, -4(%rbp)
ia9     jle .L3
```

ia4 finishes but must wait for ia3 before it can leave ROB.
ia7 has been dispatched to RS and allocated entry in ROB, and is issued.
Next cycle, ia3 will forward result to ia5 using the tag "%rob3".

ia5	1	%rob3	0	%rob4/**%rob5**	1	0
ia6	1	%rob5	0	--	1	0
ia3	1	%rob1	1	%rob2/**%rob3**	1	1
ia7	1	$1	1	%rbp	1	1



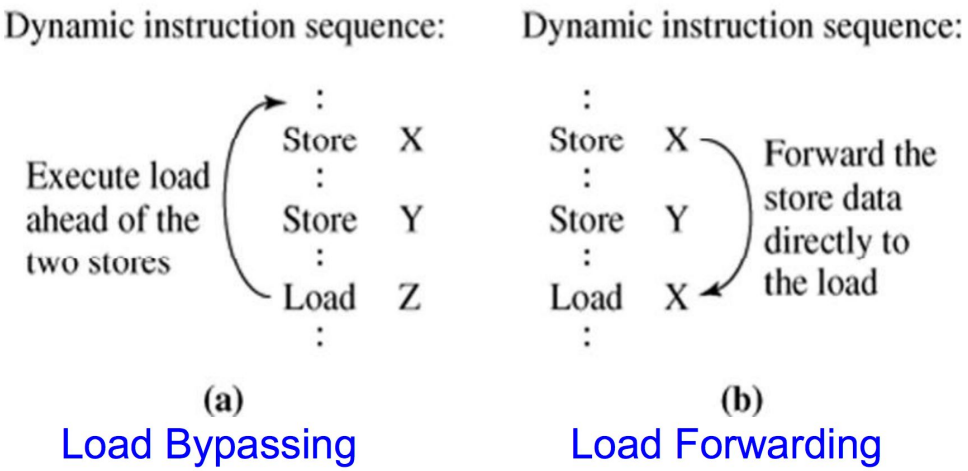
TP HP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
I									1	0	0	1	1			
F									0	0	0	1	0			
IA									ia7	ia6	ia5	ia4	ia3			
RR											rob5	rob4	rob3			
S																
V									0	0	0	1	0			

Important observations @ c4

- Register renaming removes WAR dependency between ia3 and ia4 on %xmm1, and thus allow out-of-order execution
  - ia4 finishes before ia3 once WAR is removed @ c4
- RS entries are freed earlier than ROB entries
  - RS entries are freed as soon as execution finishes (e.g. ia1@c1, ia2@c2, & ia4@c3), but ROB entry only gets freed if it is at head of the ROB (e.g. ia1@c3, ia2@c4).
  - This should help in choosing optimal RS/ROB sizes and provide analysis (last part) in part C of Arch Lab.

Recap - Load Bypassing & Forwarding



- Implemented with Load & Store Queues (LSQ)
- Detailed example in lecture slides.

Gem5 Demo

Questions

Backup Slides

Dynamic Scheduling - MIPS R10K example

- MIPS IV ISA, out-of-order superscalar RISC-based microprocessor
- Pipeline stages: Dispatch(D), Issue(S), Execute(E), Complete(C)
  - D: allocate ROB and RS entries, physical registers (PR)
  - S: insn issued once its operands are valid and ready
  - E: frees RS entry at the end of E
  - C: writes dest. register, set registers' ready bit in Map table and RS
- Data structures: Reservation Stations (RS), Reorder Buffer(ROB), Map table, Free List, Common Data Bus(CDB)
  - RS: "T", "S1", "S2" are output, source 1 and 2. "+" means ready.
  - ROB: "T" is the PR storing insn's logical output, "T_old" is prev. PR
  - Map table: arch. register to PR mapping. "+" means ready.
  - Free List: free PR's available for allocation
  - CDB: broadcasts insn output to other insn's waiting for it.

R10K - Cycle 0

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
ht								

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	no				
Mul2	no				
Ld1	no				
Ld2	no				
St1	no				
St2	no				

Map Table	
Reg	T+
f0	PR1+
f2	PR2+
f4	PR3+
r1	PR4+

Free List
PR5, PR6,PR7,PR8

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Example based on Prof. Natalie Enright-Jerger's "Dynamic Scheduling and Precise State" notes at University of Toronto.

R10K - Cycle 1

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
ht	1	L.S. 0(r1) -> f0	PR5	PR1	c1			

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	no				
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	no				
St1	no				
St2	no				

Map Table	
Reg	T+
f0	PR5
f2	PR2+
f4	PR3+
r1	PR4+

Free List
~~PR5~~, PR6,PR7,PR8

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Allocate ROB and RS entries for insn #1. Allocate new physical reg. (PR5) for f0. Remember old physical reg. mapped to f0 (PR1).

R10K - Cycle 2

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2		
t	2	MUL f0, f2 -> f4	PR6	PR3	c2			

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	no				
St1	no				
St2	no				

Map Table	
Reg	T+
f0	PR5
f2	PR2+
f4	PR6
r1	PR4+

Free List
~~PR6~~,PR7,PR8

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Insn #1 enters S since its operand is ready (PR4+).
Allocate ROB/RS entries, and PR6 for insn #2.

R10K - Cycle 3

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
t	3	S.S f4 -> 0(r1)			c3			

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR5
f2	PR2+
f4	PR6
r1	PR4+

Free List
PR7,PR8

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Stores do not allocate PR's.
Insn #2 cannot enter S as its operand (PR5) is not ready.

R10K - Cycle 4

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
t	4	SUB r1, #8 -> r1	PR7	PR4	c4			

Insn #3 cannot enter S because its operand (PR6) is not ready.

Reservation Stations					
FU	busy	op	T	S1	S2
Add	yes	SUB	PR7	PR4+	
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR5
f2	PR2+
f4	PR6
r1	PR7

Free List
~~PR7~~,PR8

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

R10K - Cycle 5

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5		
t	5	BNEZ r1, Loop			c5			

Reservation Stations					
FU	busy	op	T	S1	S2
Add	yes	SUB	PR7	PR4+	
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR5
f2	PR2+
f4	PR6
r1	PR7

Free List
PR8

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Insn #4 enters S since its operand (PR4+) is ready.
Dispatching insn #5 - will read PR7 once it is ready.

R10K - Cycle 6

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	
	5	BNEZ r1, Loop			c5			
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6			

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	yes	L.S.	PR8	PR7	
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8
f2	PR2+
f4	PR6
r1	PR7

Free List
~~PR8~~

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Insn #4 finishes X and frees its RS entry.
Branch predicted taken - dispatching insn #6 L.S. from next loop iteration.

R10K - Cycle 7

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7		
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6			

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	yes	L.S.	PR8	PR7+	
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8
f2	PR2+
f4	PR6
r1	PR7+

Free List

CDB
T
PR7

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Insn #4 completes, broadcasts result (PR7) and set ready bits in Map table and RS.
Only one issue per cycle (both insn #5 and #6 can enter S once PR7+ ready, choose insn #5).
No new insn dispatches because we ran out of PR's (empty Free List).

R10K - Cycle 8

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8		

Insn #6 is issued since its operand (PR7+) is ready.

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	yes	L.S.	PR8	PR7+	
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8
f2	PR2+
f4	PR6
r1	PR7+

Free List

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

R10K - Cycle 9

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c3+	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	c9
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8		

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	yes	L.S.	PR5	PR4+	
Ld2	yes	L.S.	PR8	PR7+	
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8
f2	PR2+
f4	PR6
r1	PR7+

Free List

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Branch completes, prediction is correct.

R10K - Cycle 10

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c10	
	2	MUL f0, f2 -> f4	PR6	PR3	c2			
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	c9
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8		

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5	PR2+
Mul2	no				
Ld1	no				
Ld2	yes	L.S.	PR8	PR7+	
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8
f2	PR2+
f4	PR6
r1	PR7+

Free List

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Insn #1 finally finishes executing (imagine cache miss, took 8 cycles)

R10K - Cycle 11

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
h	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c10	c11
	2	MUL f0, f2 -> f4	PR6	PR3	c2	c11		
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	c9
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8	c11	

Insn #1 completes and sets ready bits for PR5.
Insn #6 takes 1 cycle to finish execute (imagine cache hit).

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5+	PR2+
Mul2	no				
Ld1	no				
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8
f2	PR2+
f4	PR6
r1	PR7+

Free List

CDB
T
PR5

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

R10K - Cycle 12

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c10	c11
h	2	MUL f0, f2 -> f4	PR6	PR3	c2	c11	c12+	
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	c9
t	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8	c11	c12

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5+	PR2+
Mul2	no				
Ld1	no				
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8+
f2	PR2+
f4	PR6
r1	PR7+

Free List
PR1

CDB
T
PR8

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

Head of ROB retires, head_ptr++.
PR1 returned to Free List and release RS.
Insn #6 completes, broadcasts result, set ready bits in Map table and RS.

R10K - Cycle 13

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
	1	L.S. 0(r1) -> f0	PR5	PR1	c1	c2	c10	c11
h	2	MUL f0, f2 -> f4	PR6	PR3	c2	c11	c12+	
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	c9
	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8	c11	c12
t	7	MUL f0, f2 -> f4	PR1	PR6	c13			

Insn #7 allocates newly freed PR1

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5+	PR2+
Mul2	yes	MUL	PR1	PR8+	PR2+
Ld1	no				
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	no				

Map Table	
Reg	T+
f0	PR8+
f2	PR2+
f4	PR1
r1	PR7+

Free List
~~PR1~~

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop

R10K - Cycle 14

ROB & Insn status					Stages			
ht	#	Instruction	T	T_old	D	S	X	C
t	8	S.S. f4 -> 0(r1)			c14			
h	2	MUL f0, f2 -> f4	PR6	PR3	c2	c11	c12+	
	3	S.S f4 -> 0(r1)			c3			
	4	SUB r1, #8 -> r1	PR7	PR4	c4	c5	c6	c7
	5	BNEZ r1, Loop			c5	c7	c8	c9
	6	L.S. 0(r1) -> f0	PR8	PR5	c6	c8	c11	c12
	7	MUL f0, f2 -> f4	PR1	PR6	c13	c14		

Insn #7 both operands ready, can be issued.
Insn#8 do not need any PR.

Reservation Stations					
FU	busy	op	T	S1	S2
Add	no				
Mul1	yes	MUL	PR6	PR5+	PR2+
Mul2	yes	MUL	PR1	PR8+	PR2+
Ld1	no				
Ld2	no				
St1	yes	S.S.		PR4+	PR6
St2	yes	S.S.		PR1	PR7+

Map Table	
Reg	T+
f0	PR8+
f2	PR2+
f4	PR1
r1	PR7+

Free List

CDB
T

Loop: L.S. 0(r1) -> f0
MUL f0, f2 -> f4
S.S. f4 -> 0(r1)
SUB r1, #8 -> r1
BNEZ r1, Loop