CS 6354: Homework 1 Post-Mortem / MIPS R10000

26 September 2016

To read more...

This day's paper:

Yeager, "The MIPS R10000 Superscalar microprocessor"

Also discussed:

Homework 1 on caches

Supplementary readings:

Kanter, "Intel's Haswell CPU Microarchitecture"

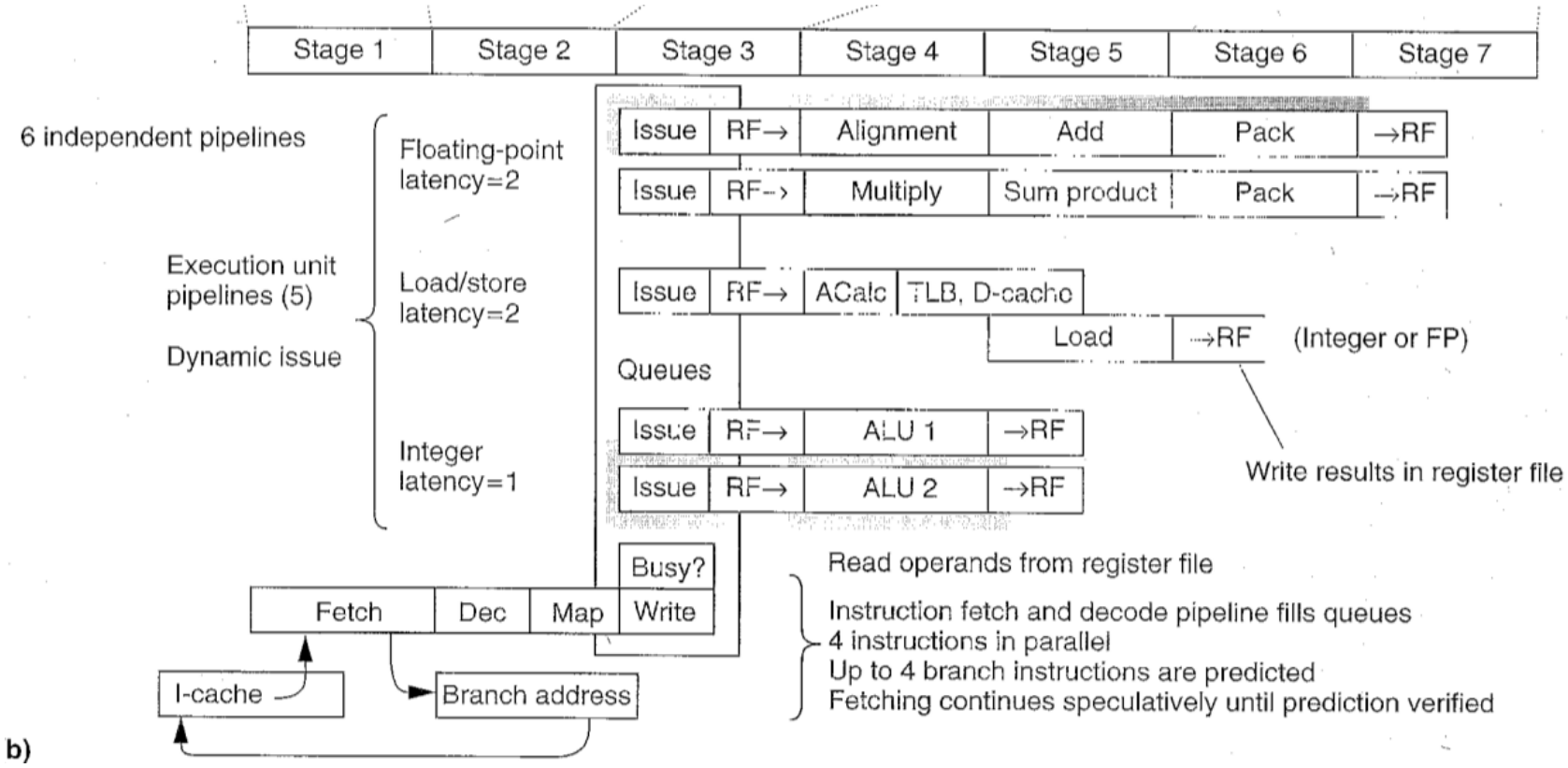
MIPS R10000: Weird names

instruction queue $\approx$ (shared) reservation station

active list $\approx$ reorder buffer

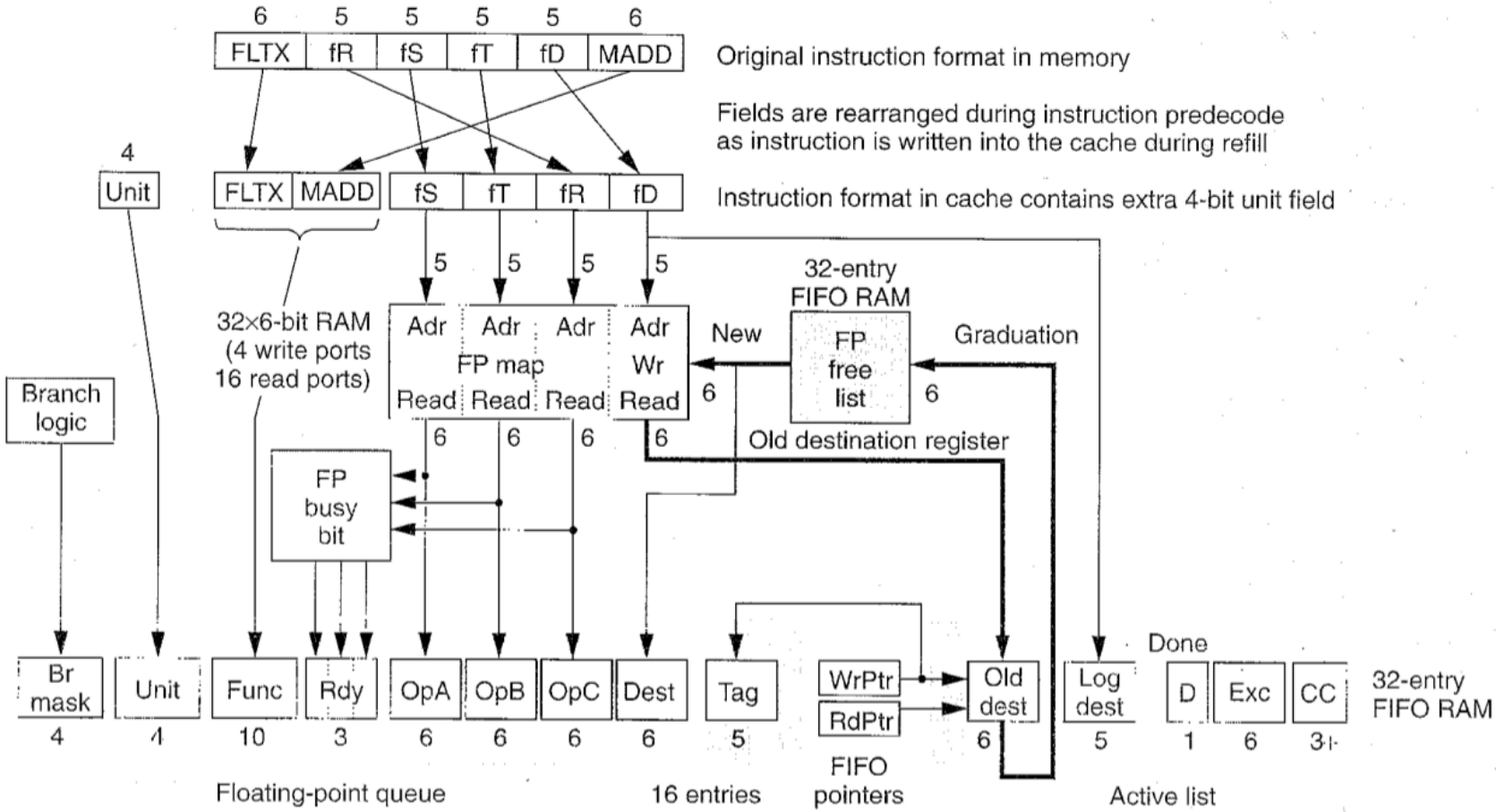
both don't store values — actually in register file

MIPS R10000: Stages



b)

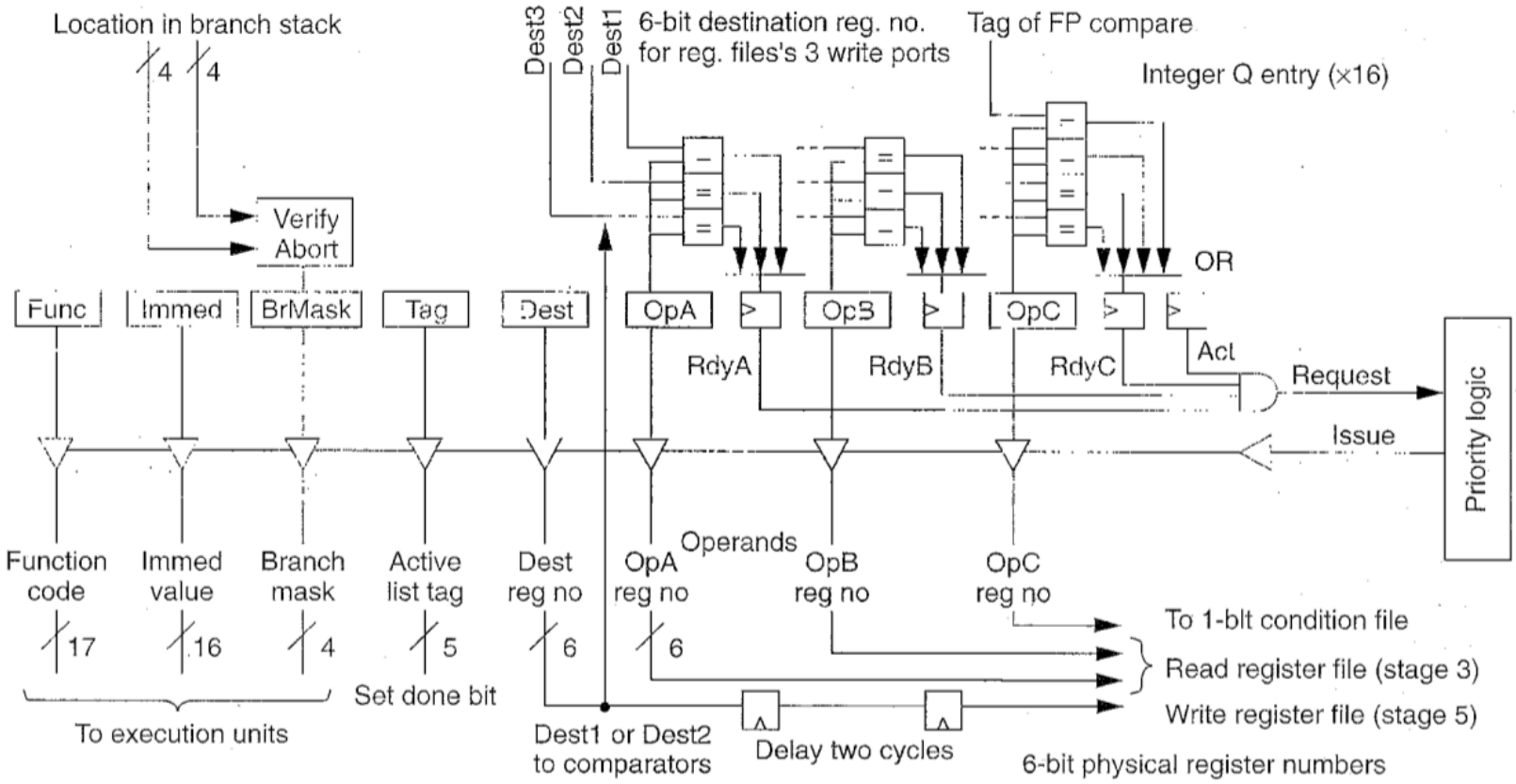
MIPS R10000: Register Renaming/Queues



MIPS R10000: Register Renaming

explicit register map data structure

MIPS R10000: Instruction Queue



MIPS R10000: Instruction Queue v. Reservation Station

shared register file

queue only tracks register numbers

metadata:

branch mask — for branch mispredicts

ready bits — local copy of busy bits

pointer to active list (ROB)

MIPS R10000: Functional Units

Table 1. Latency and repeat rates for integer instructions.

Unit	Latency (cycles)	Repeat rate (cycles)	Instruction
Either ALU	1	1	Add, subtract, logical, move Hi/Lo, trap
ALU 1	1	1	Integer branches
ALU 1	1	1	Shift
ALU 1	1	1	Conditional move
ALU 2	5/6	6	32-bit multiply
	9/10	10	64-bit multiply (to Hi/Lo registers)
ALU 2	34/35	35	32-bit divide
	66/67	67	64-bit divide
Load/store	2	1	Load integer
	—	1	Store integer

Table 2. Latency and repeat rates for floating-point instructions.

Unit	Latency (cycles)	Repeat rate (cycles)	Instruction
Add	2	1	Add, subtract, compare
Multiply	2	1	Integer branches
Divide	12	14	32-bit divide
	19	21	64-bit divide
Square root	18	20	32-bit square root
	33	35	64-bit square root
Load/store	3	1	Load floating-point value
	—	1	Store floating-point value

Moving load/stores around

program order	desired (fast) order
store X	load Z
store Y	store X
load Z	store Y

Moving load/stores around

program order	desired (fast) order
store X	load Z
store Y	store X
load Z	store Y

what if X == Z or Y == Z?

MIPS R10000: Memory requests

16 entry address queue

kept in program order

tracks dependencies (overlapping memory accesses)

special-case for two accesses to same cache set

match cache accesses against all loads

load to store forwarding

MIPS R10000: Synchronization

execute memory accesses in order

... in case other processors are listening

treat like exception if other processors are listening

LL/SC atomic increment

```
retry: // $t0 ← value
       ll $t0, value

       // $t0 < $t0 + 1
       addi $t0, $t0, 1

       // value ← $t0 if memory unchanged
       // $t0 ← 1 if stored, 0 otherwise
       sc $t0, value

       // if sc unsuccessful, goto retry
       beqz $t0, retry
       nop                  // (delay slot)
```

MIPS R10000: Weird Tricks

predecoding in instruction cache — opcode preprocessed

instruction cache specialized for unaligned accesses within a block

multibanked data cache — half the sets in one cache, half in another

core storage (approx sizes)

data — approx 8KB

- register files — 8192 bits

metadata — approx 4KB

- register map tables — 390 bits
- free list — 192 bits
- active list — 672 bits
- busy bits: — 128 bits
- instruction queues — 1600 bits
- address queue — 1232 bits

SGI's workload

graphics

lots of floating point

big images

evolution of modern processors

	MIPS R10000 (1996)	Intel Haswell (2013)
fetch/cycle	4 instructions	5 instructions
reorder buffer	32 entry	192 entry
instruction queue	16 int + 16 FP + 16 mem	60 unified
execute/cycle	2 int + 2 FP	2 int/FP + 2 int
memory/cycle	1 load or store	2 load + 1 store
operand width	32 bit	32 bit to 256 bit
L1 cache	32K I, 32K D	32K I, 32K D
L2 cache	off-chip	256K
L3 cache	none	1+MB
L1 TLB	64 entry	64 entry D, 64 entry I
L2 TLB	none	1024 entry
predecoding	in I-cache	micro-op cache
branch pred.	local, 512 entry	???
cores/package	1	2–18

evolution of modern processors

	MIPS R10000 (1996)	Intel Haswell (2013)
fetch/cycle	4 instructions	5 instructions
reorder buffer	32 entry	192 entry
instruction queue	16 int + 16 FP + 16 mem	60 unified
execute/cycle	2 int + 2 FP	2 int/FP + 2 int
memory/cycle	1 load or store	2 load + 1 store
operand width	32 bit	32 bit to 256 bit
L1 cache	32K I, 32K D	32K I, 32K D
L2 cache	off-chip	256K
L3 cache	none	1+MB
L1 TLB	64 entry	64 entry D, 64 entry I
L2 TLB	none	1024 entry
predecoding	in I-cache	micro-op cache
branch pred.	local, 512 entry	???
cores/package	1	2–18

evolution of modern processors

	MIPS R10000 (1996)	Intel Haswell (2013)
fetch/cycle	4 instructions	5 instructions
reorder buffer	32 entry	192 entry
instruction queue	16 int + 16 FP + 16 mem	60 unified
execute/cycle	2 int + 2 FP	2 int/FP + 2 int
memory/cycle	1 load or store	2 load + 1 store
operand width	32 bit	32 bit to 256 bit
L1 cache	32K I, 32K D	32K I, 32K D
L2 cache	off-chip	256K
L3 cache	none	1+MB
L1 TLB	64 entry	64 entry D, 64 entry I
L2 TLB	none	1024 entry
predecoding	in I-cache	micro-op cache
branch pred.	local, 512 entry	???
cores/package	1	2–18

Micro-ops

complex instruction encodings don't allow pre-decode trick

complex insturctions can't go to a single functional unit

trick: split into micro-ops

extra decoding step

Intel Haswell: cache for micro-ops

Homework 1: Rubric

140 points total: For each thing benchmarked:

- 5 points: benchmark description, including how to read results
- 5 points: raw results and code are included, match description, interpreted plausibly

10 points: tested system described, report parts clear

Homework 1: General Concerns: Benchmarking Discipline

how consistent are your measurements?

is that really an increase?

be honest

Homework 1: General Concerns: Latency v. Bandwidth

bandwidths much better than latencies (everywhere)

memory system relies on overlapping many memory accesses

measuring sizes? better to measure latency

better to avoid prefetching — e.g. random access pattern, pointer chasing

Next Time: SMT

multiple threads on one core

later: multiple processors/cores

Definition: Thread

stream of program execution

own registers

own program counter (current instruction pointer)

may or may not share memory

appears to execute at same time as other threads

Multithreading

```
thread_one_func(int offset) {
    for (int i = 0; i < N / 2; ++i)
        sum1 += array[offset + i];
}
thread_two_func() {
    for (int i = N / 2; i < N; ++i)
        sum2 += array[i];
}
compute_sum() {
    thread_one = thread_create(thread_one_func);
    thread_two = thread_create(thread_two_func);
    wait_for_thread(thread_one);
    wait_for_thread(thread_two);
    sum = sum1 + sum2;
}
```

Different Parallelism

instruction-level parallelism

- sequential sequence of instructions
- not actually sequential
- transparent to programmer

next up: thread-level parallelism

- multiple sequential sequences of instructions
- run in parallel (apparently)
- exposed to programmer

later: vectorization

- sequential sequence of instructions
- that each does multiple copies of the same thing
- exposed to programmer

Flynn's Taxonomy

	Single instruction	Multiple instruction
Single data	serial	???
Multiple data	vectors	threads