

Review of Von Neumann Architectures

A little history... programs

- **Stored program model has been around for a long time...**

First Draft of a Report
on the EDVAC

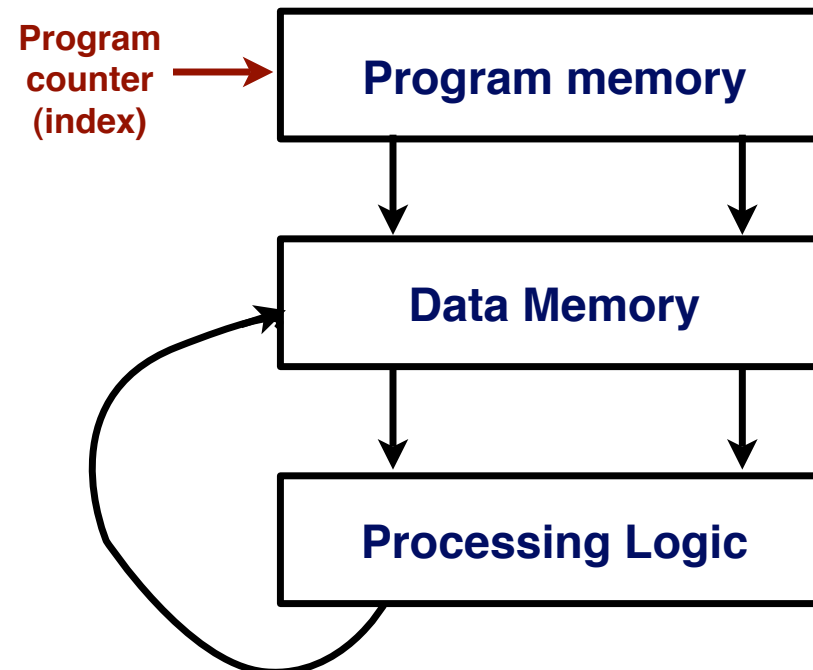
by
John von Neumann

Contract No. W-670-ORD-4926

Between the
United States Army Ordnance Department
and the
University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

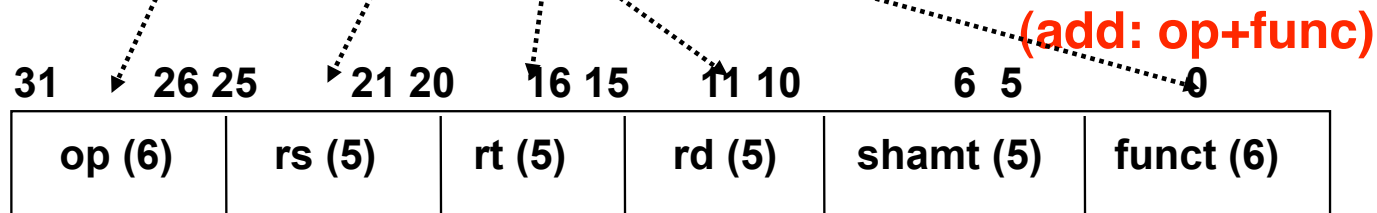


R-Type: Assembly and Machine Format ₃

Compiler translates HLL to instructions which reduce to 1s, 0s

- R-type: All operands are in registers

Assembly: **add \$9, \$7, \$8 # add rd, rs, rt: RF[rd] = RF[rs]+RF[rt]**



Machine:

B:	000000	00111	01000	01001	xxxxx	100000
D:	0	7	8	9	x	32

R-type Instructions

- ❑ All instructions have 3 operands
- ❑ All operands must be registers
- ❑ Operand order is fixed (destination first)
- ❑ Example:

C code: `A = B - C;`

(Assume that A, B, C are stored in registers s0, s1, s2.)

MIPS code: `sub $s0, $s1, $s2`

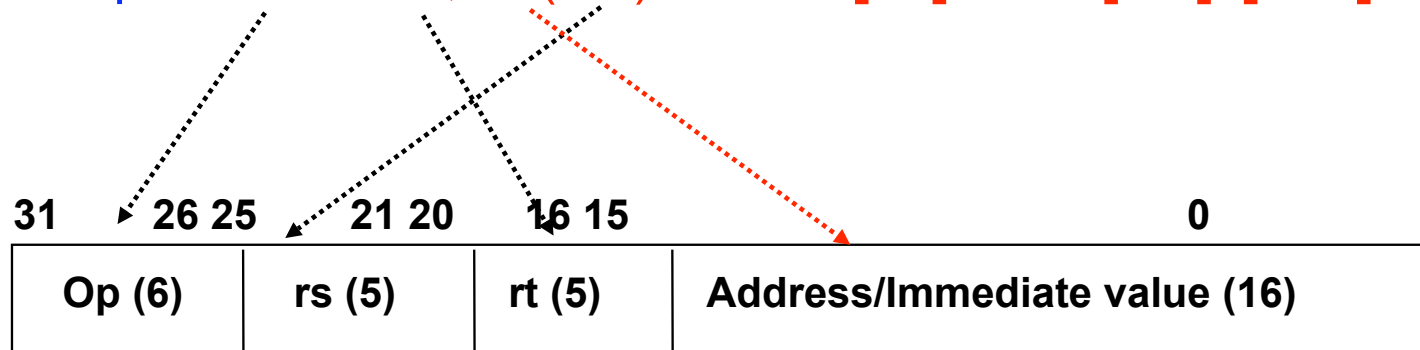
Machine code: `000000 10001 10010 10000 xxxxx 100010`

- ❑ Other R-type instructions
 - `addu, mult, and, or, sll, srl, ...`

I-Type Instructions: Another Example

- I-type: One operand is an immediate value and others are in registers

Example: **lw** \$s3, 32(\$t0) # RF[19] = DM[RF[8]+32]

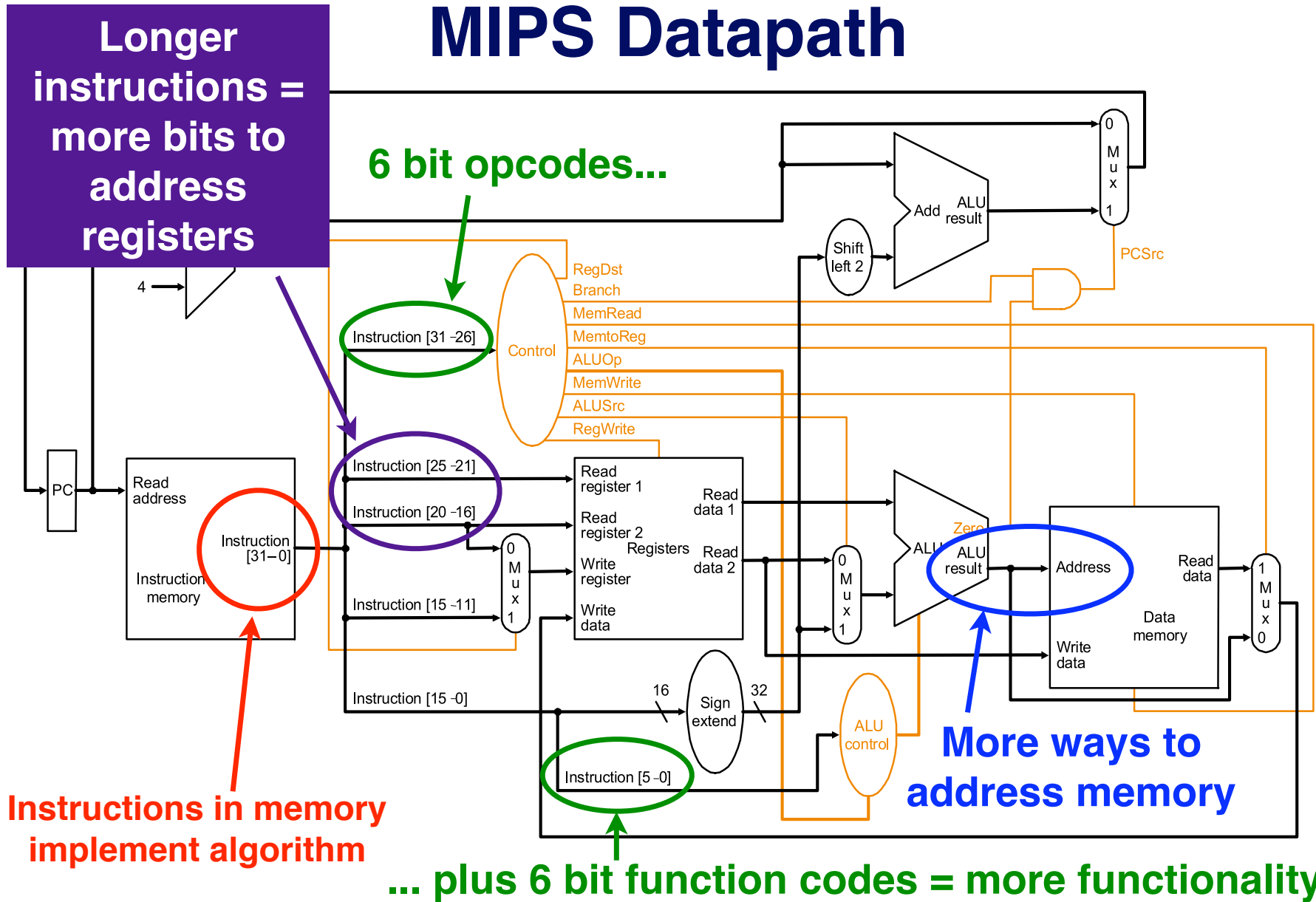


B: 100011 01000 10011 0000000000100000

D: 35 8 19 32

How about load the next word in memory?

MIPS Datapath

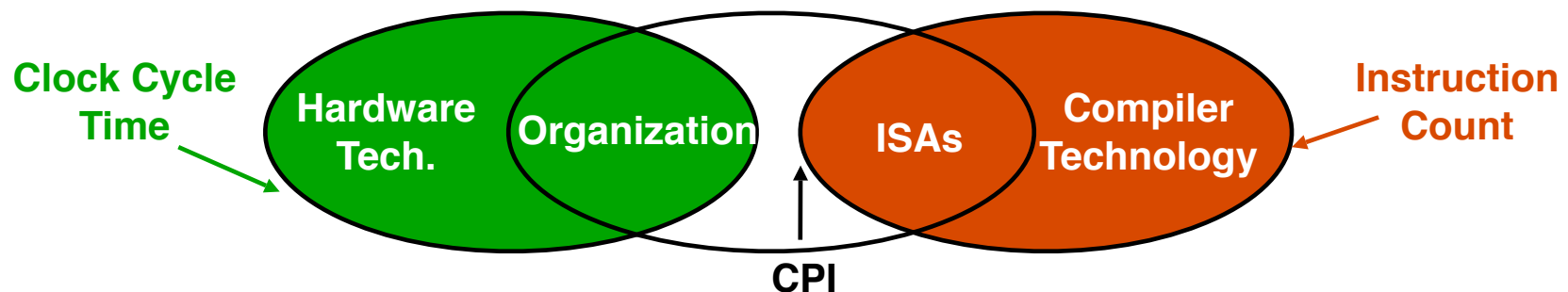


An important idea...

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

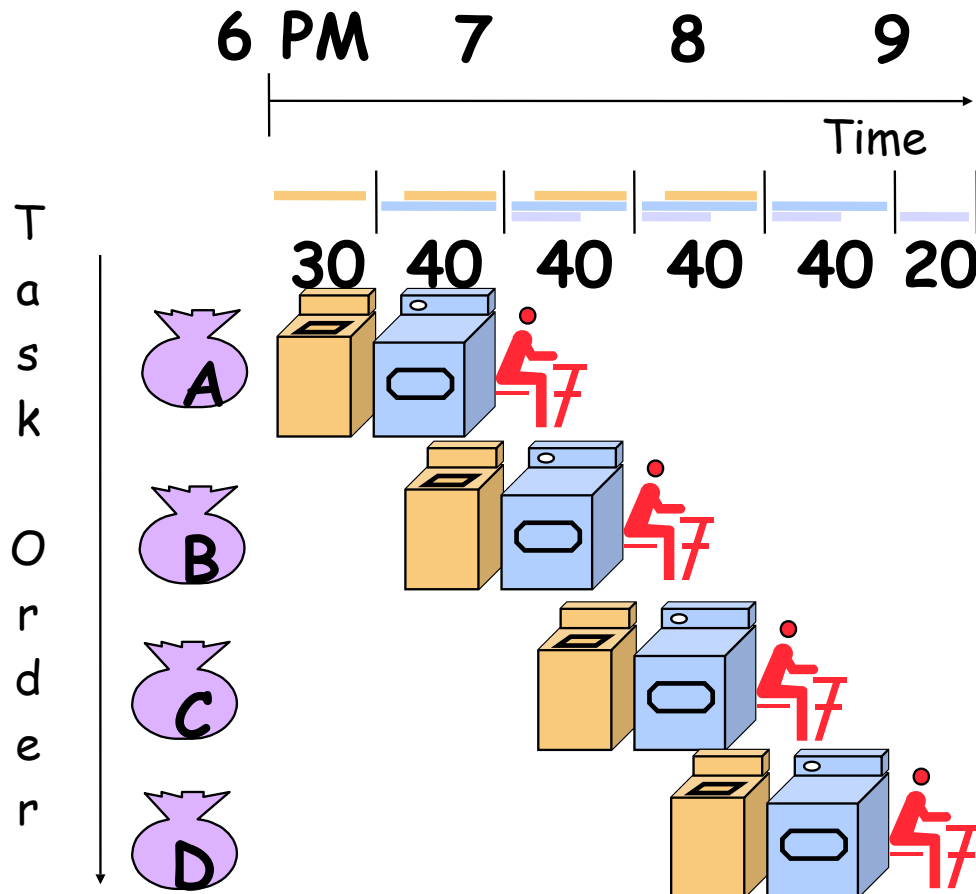
A common denominator

- We can see CPU performance dependent on:
 - **Clock rate, CPI, and instruction count**
- CPU time is directly proportional to all 3:
 - Therefore an \times % improvement in any one variable leads to an \times % improvement in CPU performance
- But, everything usually affects everything:



For new switches, time is the best benchmark;
 We can calculate speed, power for general purpose approach
 If new switch better, get win

Pipelining Lessons (laundry example)



- **Multiple** tasks operating simultaneously
- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- Pipeline rate limited by **slowest** pipeline stage
- Potential speedup = **Number pipe stages**
- Unbalanced lengths of pipe stages reduces speedup
- Also, need time to "**fill**" and "**drain**" the pipeline.

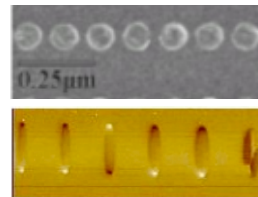
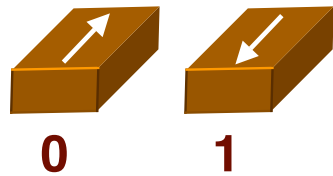
Pipelineing can cost overhead. But what if free?
What if really deep?

Example: Nanomagnetics

Schematic

Experimental

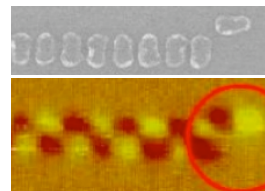
Device



R. Cowburn, M. Welland, "Room temperature magnetic quantum cellular automata," *Science* **287**, 1466, 2000

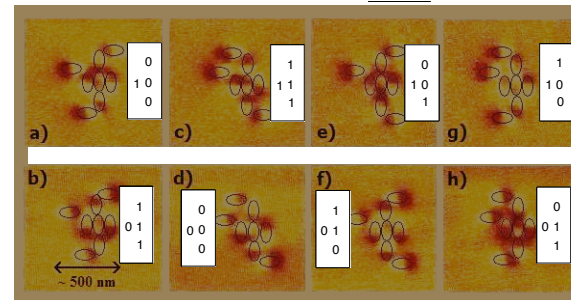
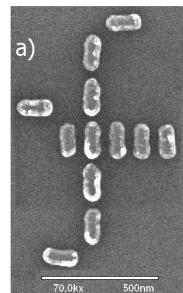
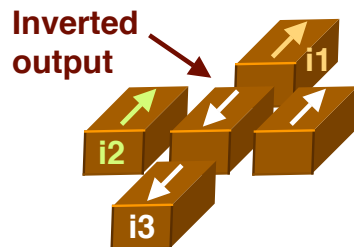
A. Imre, "Experimental Study of Nanomagnets for Magnetic QCA Logic Applications," U. of Notre Dame, Ph.D. Dissertation.

Wire



A. Imre, et. al., "Majority logic gate for Magnetic Quantum-Dot Cellular Automata," *Science*, vol. 311, No. 5758, pp. 205–208, January 13, 2006.

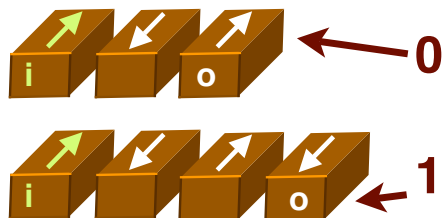
Gate



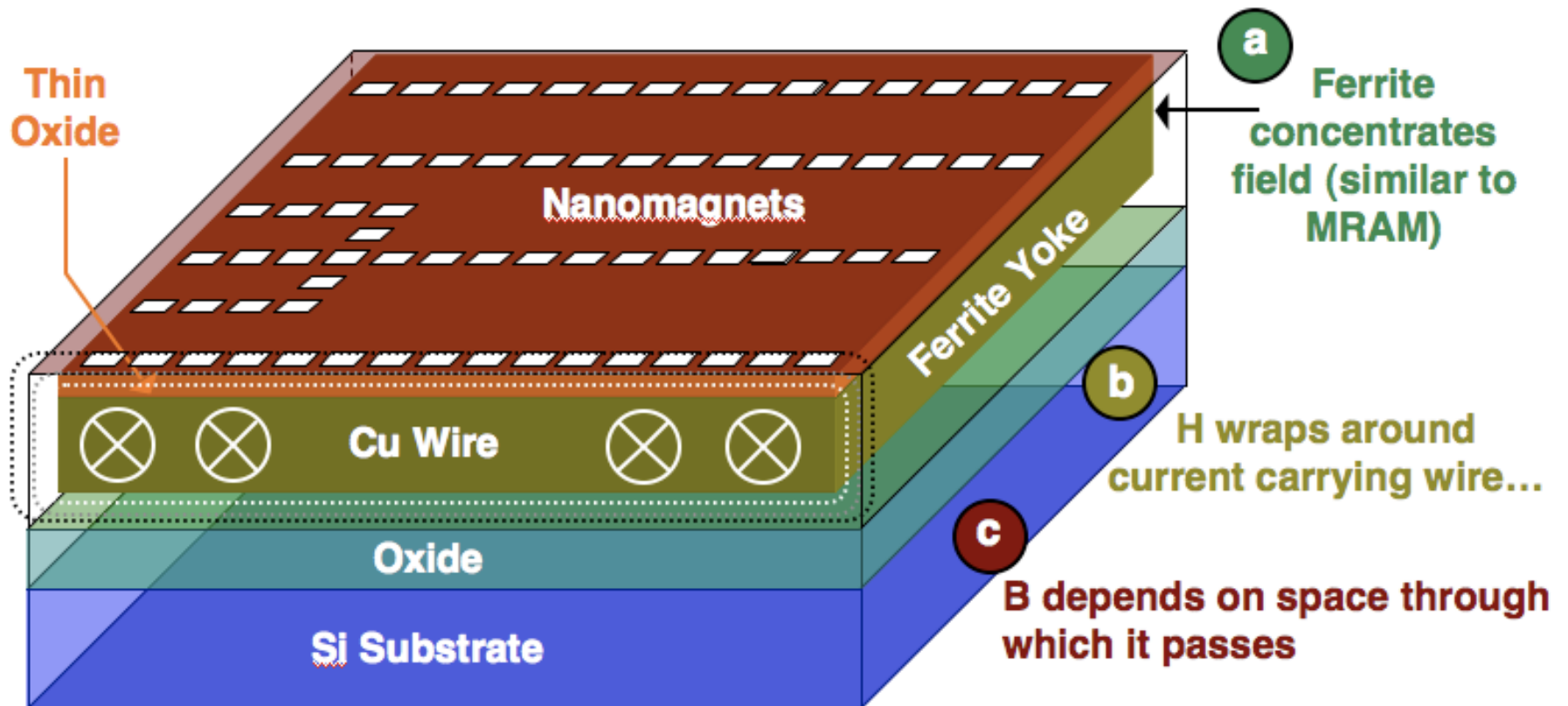
A. Imre, et. al. "Magnetic Logic Devices Based on Field-Coupled Nanomagnets," *NanoGiga* 2007.

A. Imre, et. al., "Majority logic gate for Magnetic Quantum-Dot Cellular Automata," *Science*, vol. 311, No. 5758, pp. 205–208, January 13, 2006.

Inverter

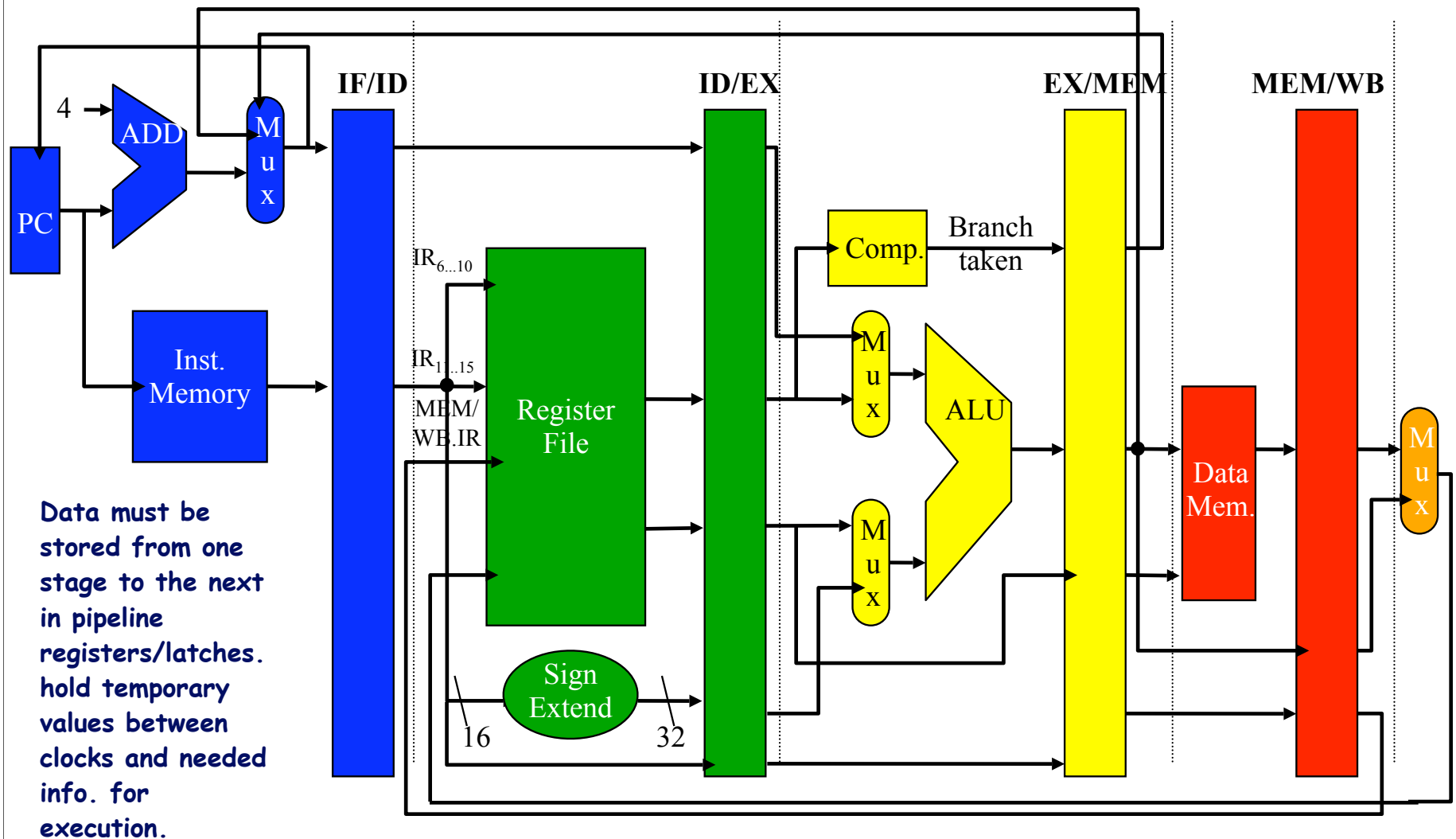


Example: Nanomagnetics

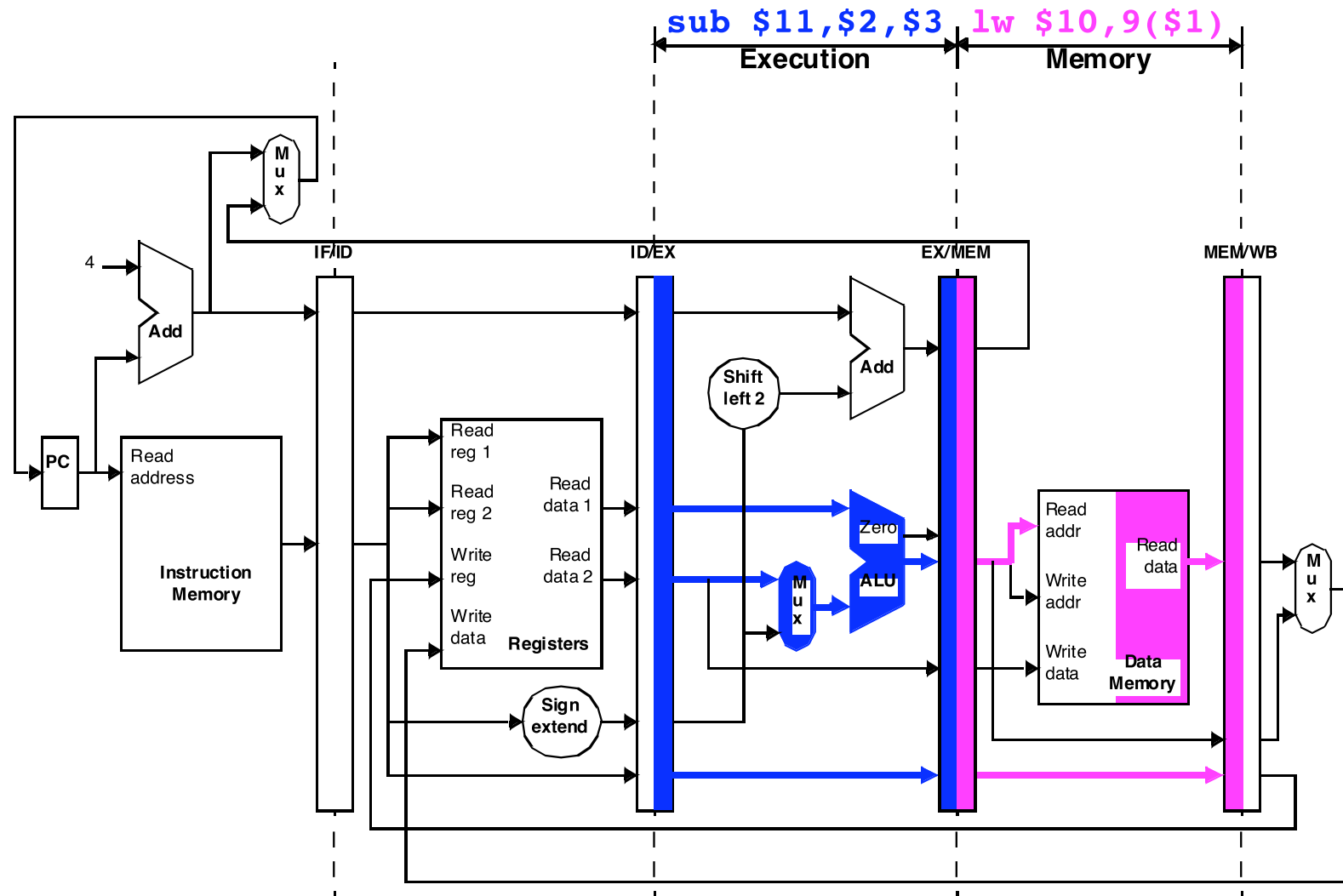


1 wire controls 1000s of magnets

The “new look” dataflow



Single-cycle diagrams: cycle 4

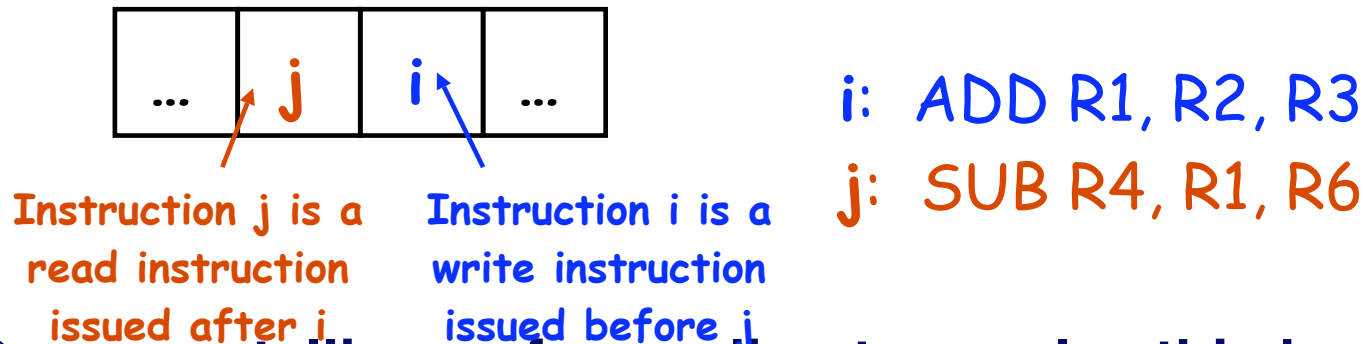


Data hazard specifics

- **There are actually 3 different kinds of data hazards!**
 - **Read After Write (RAW)**
 - **Write After Write (WAW)**
 - **Write After Read (WAR)**
- **We'll discuss/illustrate each on forthcoming slides. However, 1st a note on convention.**
 - **Discussion of hazards will use generic instructions i & j .**
 - **i is always issued before j .**
 - **Thus, i will always be further along in pipeline than j .**
- **With an in-order issue/in-order completion machine, we're not as concerned with WAW, WAR**

Read after write (RAW) hazards

- With RAW hazard, instruction j tries to read a source operand before instruction i writes it.
- Thus, j would incorrectly receive an old or incorrect value
- Graphically/Example:



- Can use stalling or forwarding to resolve this hazard

Branch/Control Hazards

- So far, we've limited discussion of hazards to:
 - Arithmetic/logic operations
 - Data transfers
- Also need to consider hazards involving branches:
 - Example:
 - 40: beq \$1, \$3, \$28 # (\$28 gives address 72)
 - 44: and \$12, \$2, \$5
 - 48: or \$13, \$6, \$2
 - 52: add \$14, \$2, \$2
 - 72: lw \$4, 50(\$7)
- How long will it take before the branch decision takes effect?
 - What happens in the meantime?

Pipelining and ILP

- **Pipelining provides for some instruction level parallelism**
 - (multiple instructions executing at the same time)
- **Hazards hurt ILP**
 - (sometimes we have to stall the pipeline and wait b/c of instruction/data dependencies)
- **Dynamic scheduling (next topic) might help...**

Dynamic Scheduling: Motivation

	1	2	3	4	5	6	7	8	9	10
<code>divf f0, f2, f4</code>	F	D	E/	E/	E/	E/	W			
<code>addf f6, f0, f2</code>		F	D	d*	d*	d*	E+	E+	W	
<code>mulf f8, f2, f4</code>			F	p*	p*	p*	D	E*	E*	W

- cycle4: `addf` stalls due to *RAW hazard*
 - OK, fundamental problem
- also cycle4: `mulf` stalls due to *pipeline hazard* (`addf` stalls)
 - why? `mulf` can't proceed into ID because `addf` is there
 - but that's the only reason \Rightarrow not good enough!
- why can't we decode `mulf` in cycle 4 and execute it in c5?
 - no fundamental reason why we can't do this!

Scheduling

scheduling: re-arranging instructions to maximize performance

- requires knowledge about structure of processor
- requires knowledge about latencies and dependences

two options for who should schedule instructions

- static scheduling: by compiler
- dynamic scheduling: by hardware

Scheduling


- **Finds instructions to execute in each cycle**
 - **Static (in-order) scheduling:**
looks only at the next instruction
 - **Dynamic (out-of-order) scheduling:**
looks at a “window” of instructions
- **How many instructions are we looking for?**
 - **3-4 is typical today, 8 is in the works**
 - **A CPU that can ideally do N instrs per cycle is called “N-way superscalar”, “N-issue superscalar”, or simply “N-way” or “N-issue”.**

Static Scheduling

- **Cycle 1**
 - **Start I1.**
 - **Can we also start I2? No.**
- **Cycle 2**
 - **Start I2.**
 - **Can we also start I3? Yes.**
 - **Can we also start I4? No.**
- **If the next instruction can not start, stops looking for things to do in this cycle!**

Program code

```
I1: ADD R1, R2, R3
I2: SUB R4, R1, R5
I3: AND R6, R1, R7
I4: OR R8, R2, R6
I5: XOR R10, R2, R11
```



Dynamic Scheduling


- **Cycle 1**
 - Operands ready? I1, I5.
 - Start I1, I5.

- **Cycle 2**
 - Operands ready? I2, I3.
 - Start I2, I3.

- **Window size (W):**
how many instructions ahead do we look.
 - Do not confuse with “issue width” (N).
 - E.g. a 4-issue out-of-order processor can have a 128-entry window (it can look at the next 128 instructions).

Program code

```
I1: ADD R1, R2, R3
I2: SUB R4, R1, R5
I3: AND R6, R1, R7
I4: OR R8, R2, R6
I5: XOR R10, R2, R11
```



Register Renaming

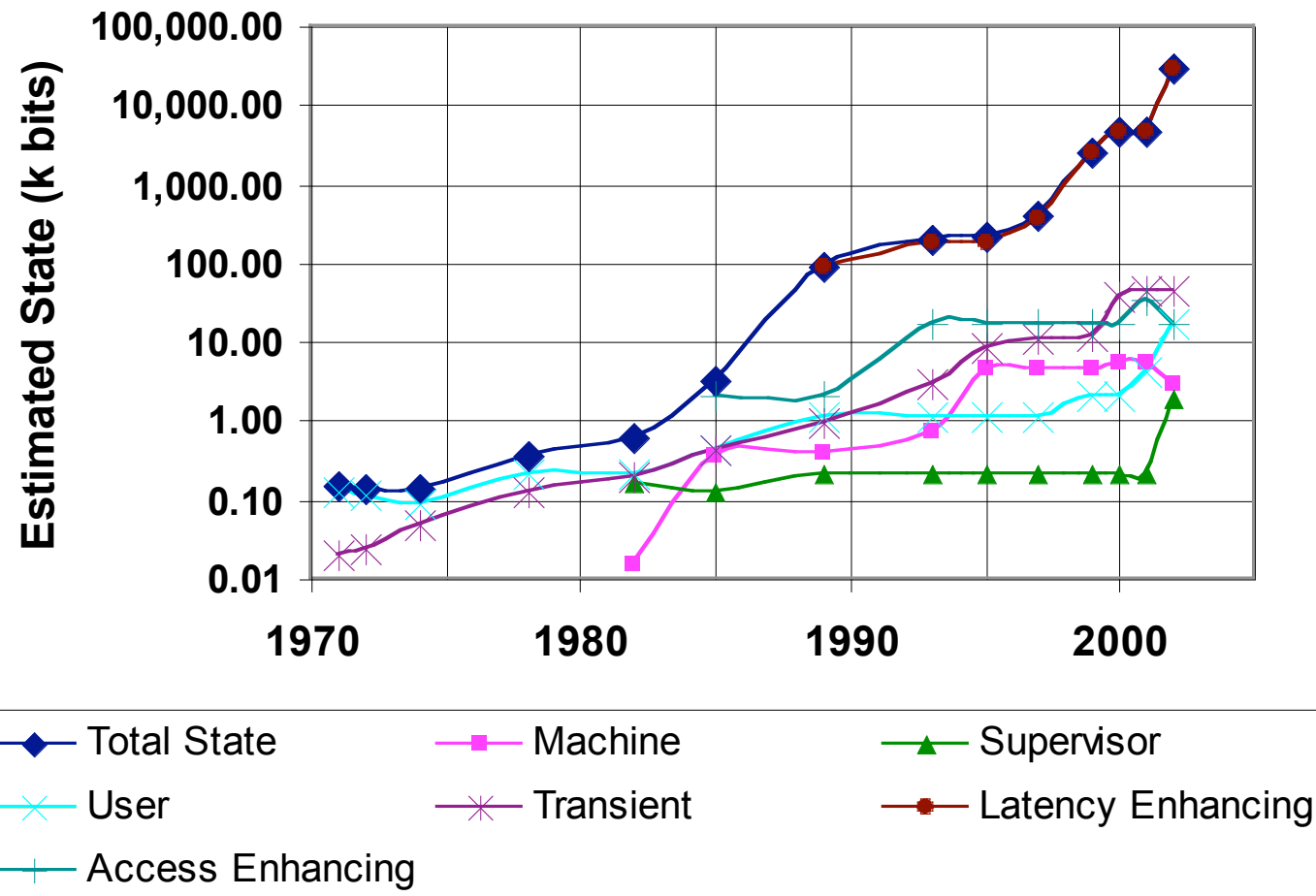
- **Solution: give I3 some other some other name (e.g. S) for the value it produces.**
- **But I4 uses that value, so we must also change that to S...**
- **In fact, all uses of R5 from I3 to the next instruction that writes to R5 again must now be changed to S!**
- **We get rid of output dependences in the same way: change R2 in I5 (and subsequent instrs) to T.**

Program code

```
I1: ADD R1, R2, R3
I2: SUB R2, R1, R5
I3: AND R5, R11, R7
I4: OR R8, R6, R2
I5: XOR R2, R4, R11
```

Overhead of dynamic scheduling

- Need excess state that keeps track of renames, etc.

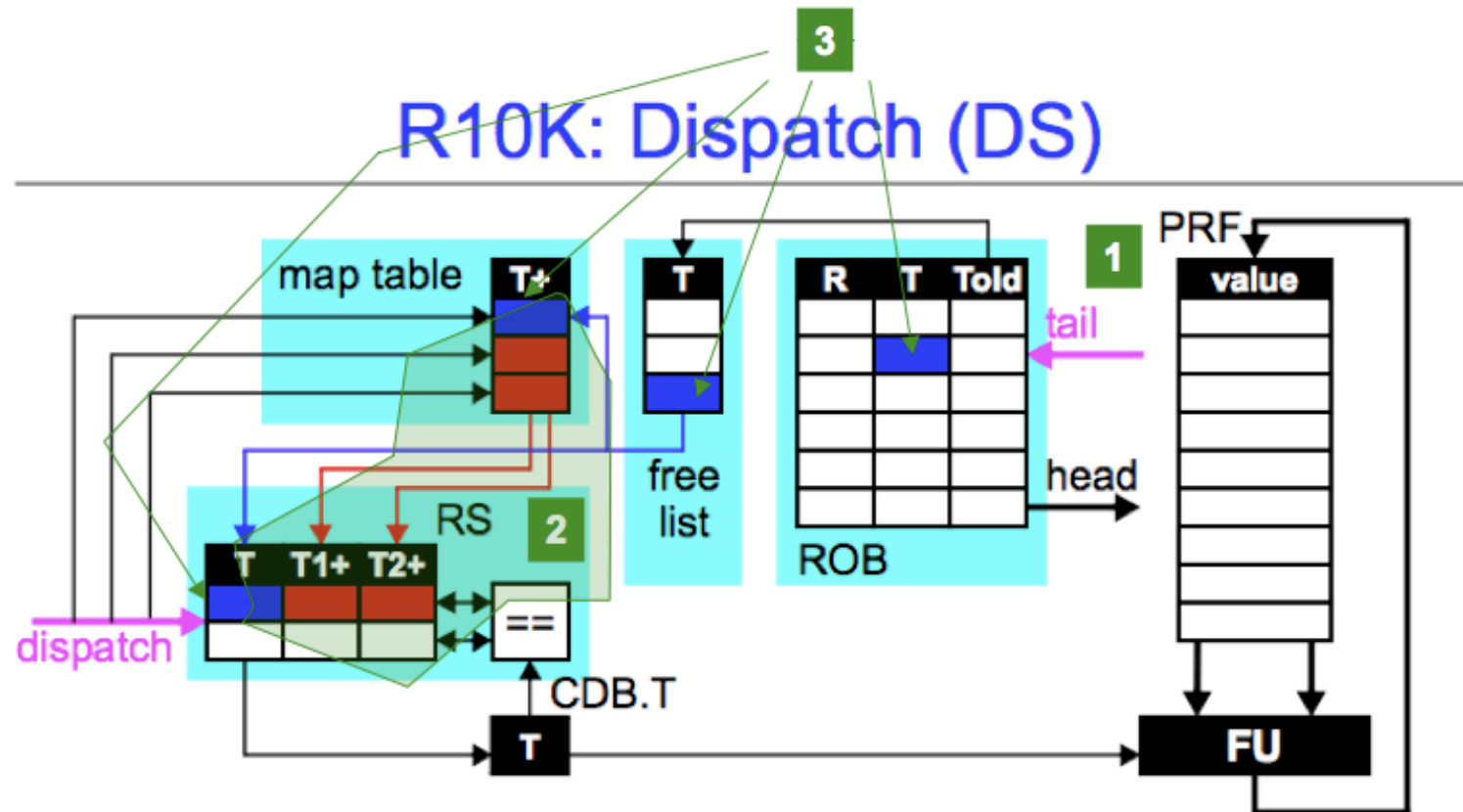


R10K Pipeline

same pipeline structure: IF, *DS*, IS, EX, *CM*, *RT* Red = different than before

- *DS (dispatch)*
 - (RS or ROB or MOB full or no physical registers) ? (stall) : If structural hazard
 - (allocate RS and ROB entries AND physical register) If no structural hazard
- *IS (issue)*
 - (read physical registers)
- *CM (completion)*
 - (writeback destination register, mark ROB entry complete)
- *RT (retire, commit, graduate)*
 - (ROB head not complete) ? (stall) :
 - {if store then write MOB head to D\$, handle any exceptions, free ROB/MOB entries, free previous physical register}

R10K: Dispatch (DS)



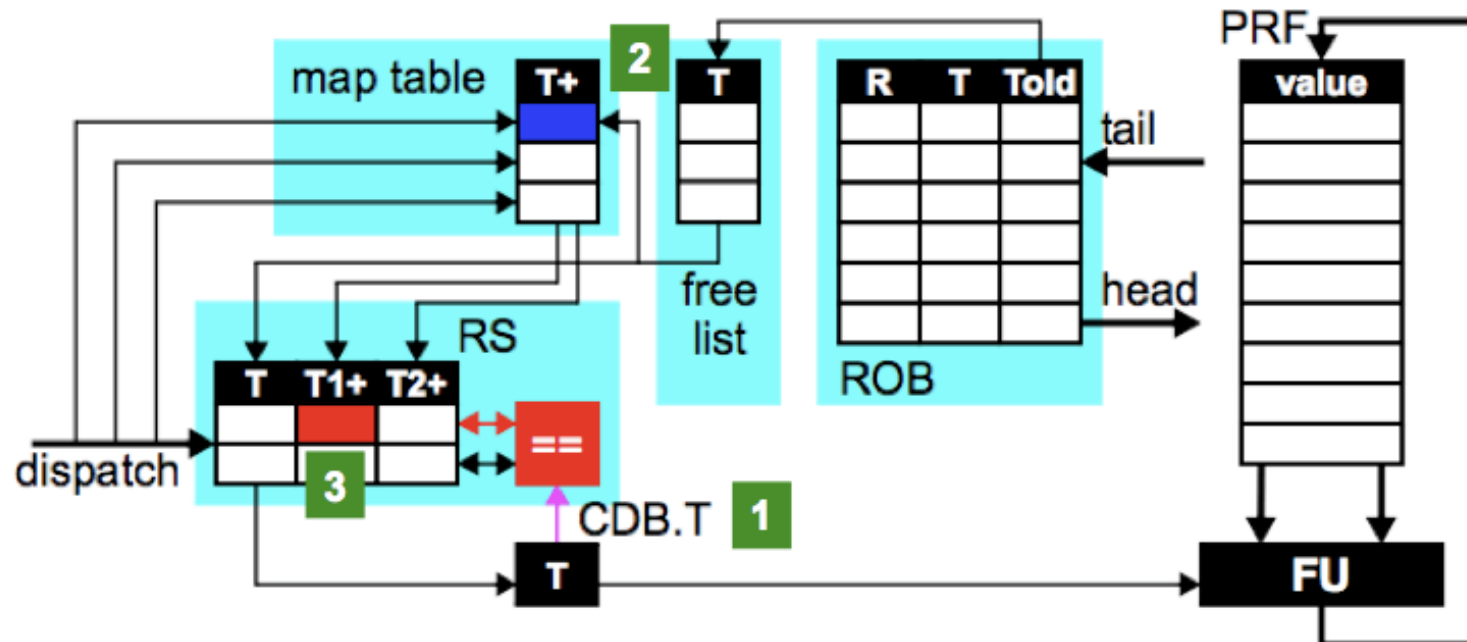
- stall if no free RS, ROB, MOB or physical register (preg)

1 • allocate RS and ROB entry

2 • read physical register tags for input registers, store in RS

3 • allocate new preg for output, set in RS, ROB and map table

R10K: Complete (CM)



- wait for free CDB

1 • broadcast tag on CDB.T

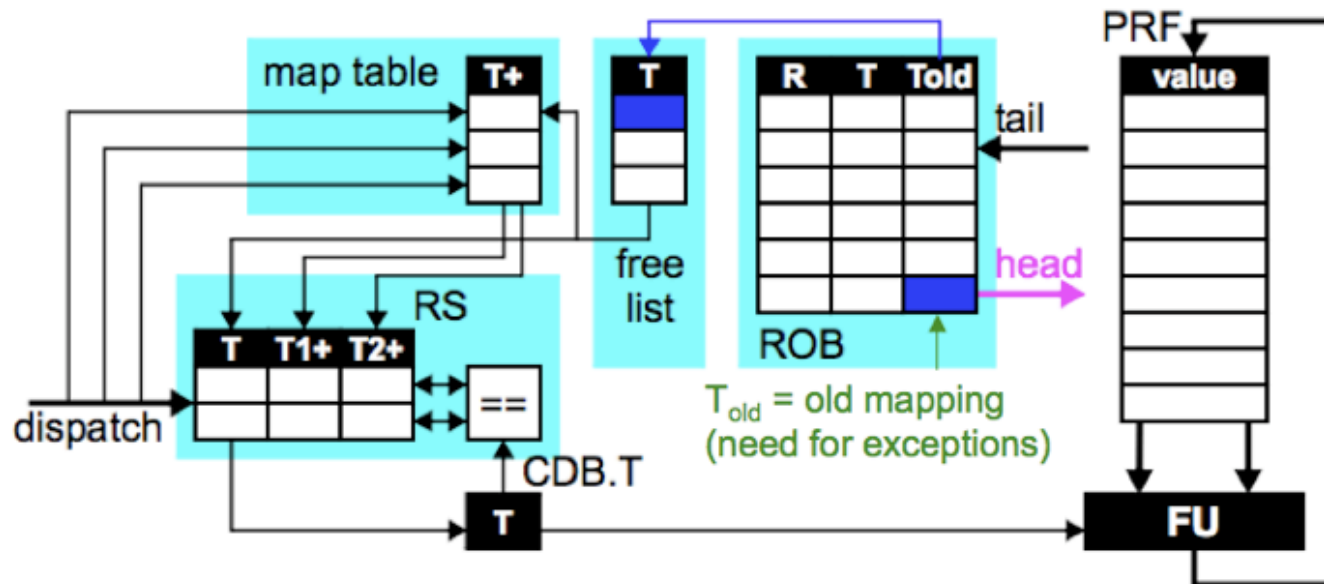
2 • set instruction's output register ready bit in map table

3 • set ready bits for matching input tags in RS

Therefore, if Add R1, R1, R1 -- R1 initially T7, R1 renamed to T3

Here, $T_{old} = T7$, $T = T3$

R10K: Retire (RT)



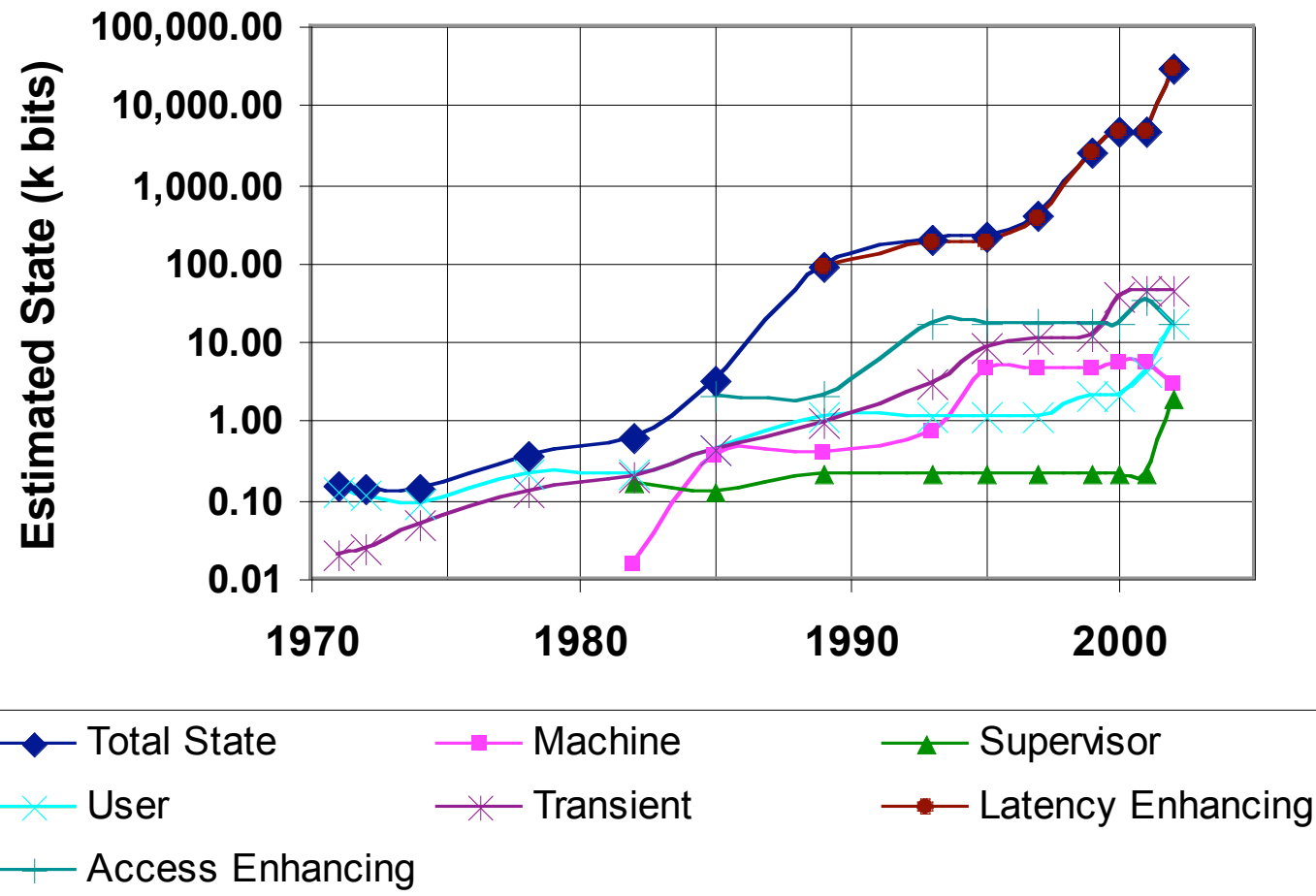
- stall until instruction at ROB head is complete

- return T_{old} of ROB head to free list
- free ROB head entry

With above example, everything waiting on T7 has value -- as we're at head of ROB
R1 now = T3

Overhead of dynamic scheduling

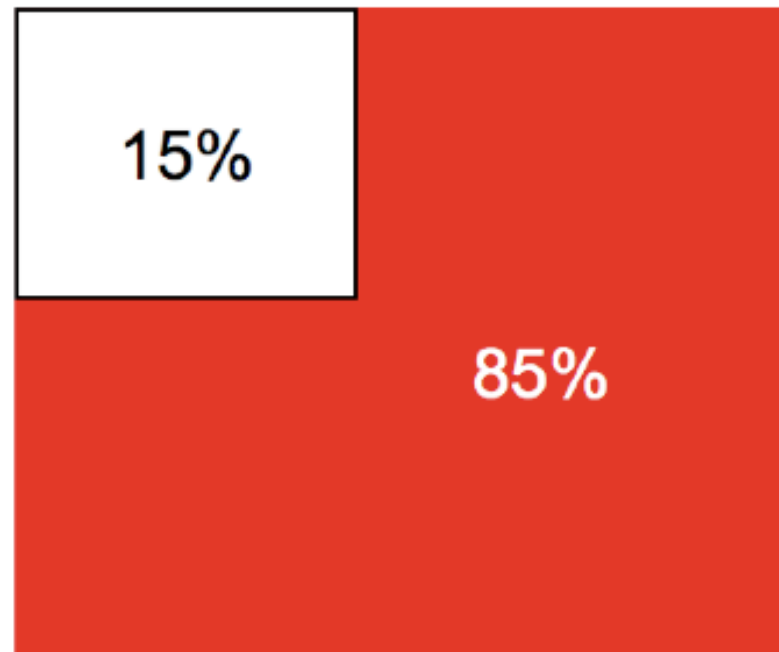
- Need excess state that keeps track of renames, etc.



Question?

- How much of a chip is “memory”?
 - 10%
 - 25%
 - 50%
 - 75%
 - 85%

Some Perspective

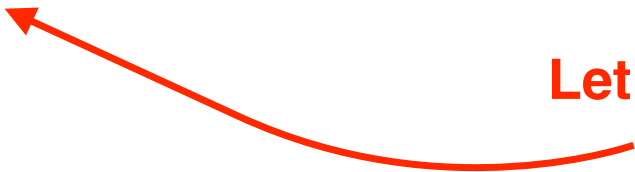


State also comes in
form of memory

If I say “Memory” what do you think of?

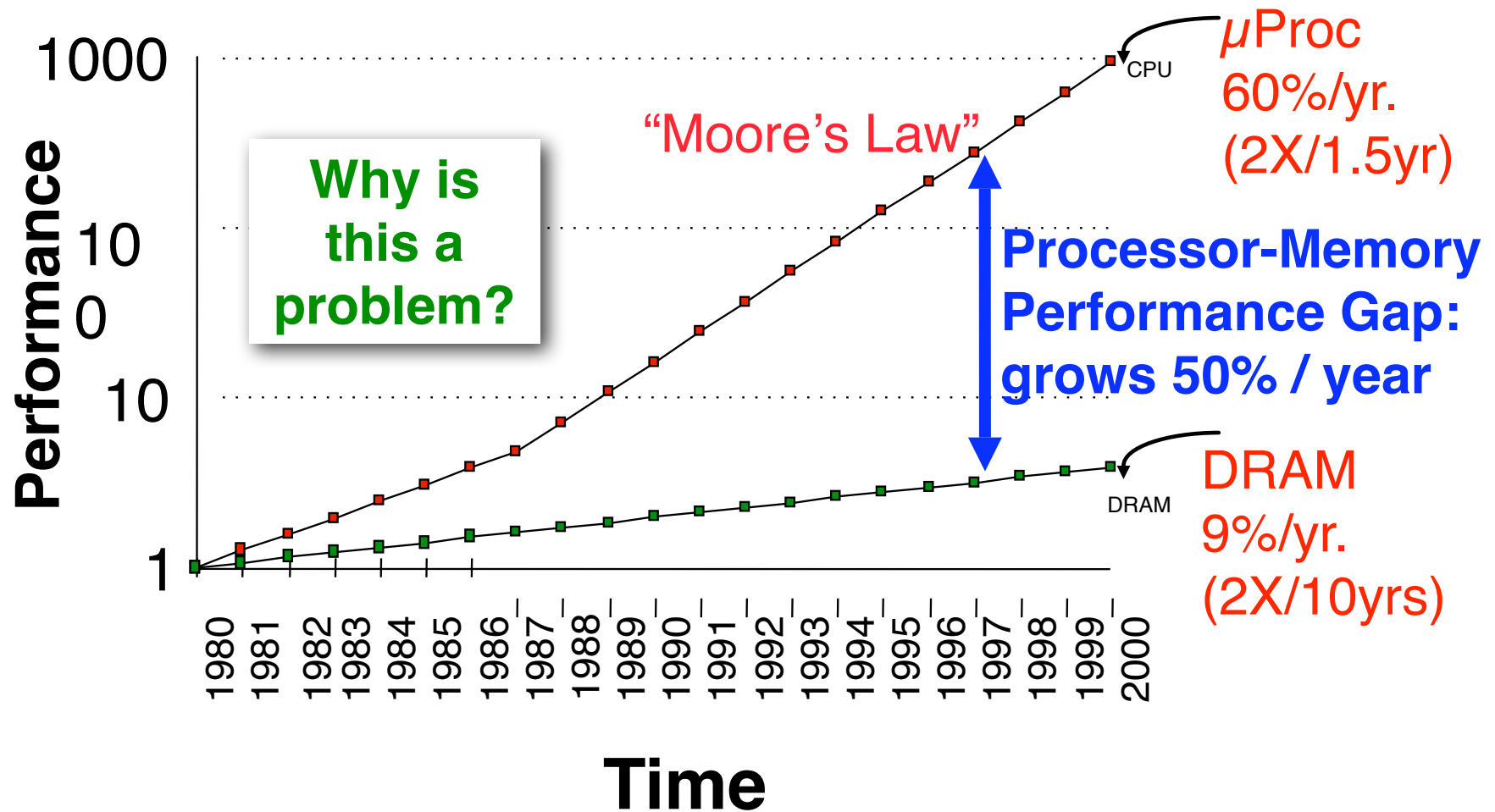
- **Memory Comes in Many Flavors**
 - **SRAM (Static Random Access Memory)**
 - **DRAM (Dynamic Random Access Memory)**
 - **ROM, Flash, etc.**
 - **Disks, Tapes, etc.**
- **Difference in speed, price and “size”**
 - **Fast is small and/or expensive**
 - **Large is slow and/or expensive**
- **The search is on for a “universal memory”**
 - **What’s a “universal memory”**
 - **Fast and non-volatile.**
 - **May be MRAM, PCRAM, etc. etc.**

**Let’s start with
DRAM.
Its generally the
largest piece of
RAM.**



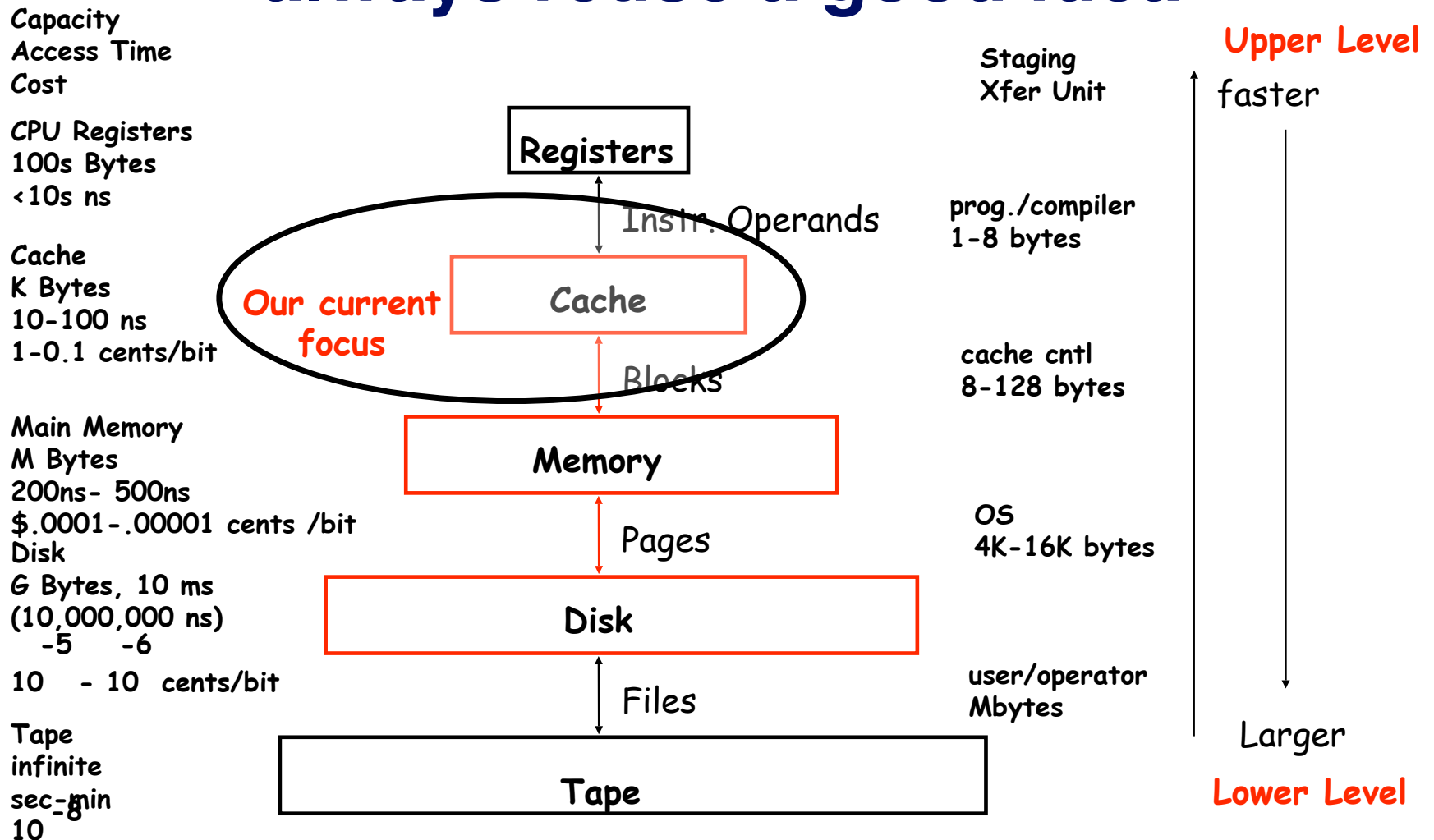
Is there a problem with DRAM?

Processor-DRAM Memory Gap (latency)



The Full Memory Hierarchy

“always reuse a good idea”



Where can a block be placed in a cache?

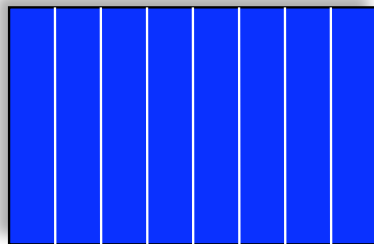
- **3 schemes for block placement in a cache:**
 - **Direct mapped cache:**
 - Block (or data to be stored) can go to only 1 place in cache
 - Usually: (Block address) MOD (# of blocks in the cache)
 - **Fully associative cache:**
 - Block can be placed anywhere in cache
 - **Set associative cache:**
 - “Set” = a group of blocks in the cache
 - Block mapped onto a set & then block can be placed anywhere within that set
 - Usually: (Block address) MOD (# of sets in the cache)
 - If n blocks, we call it n-way set associative

Where can a block be placed in a cache?

Cache:

Fully Associative

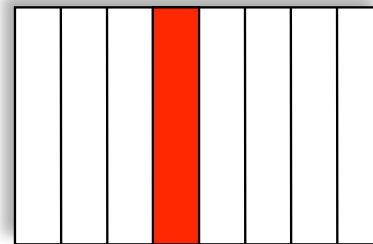
1 2 3 4 5 6 7 8



Block 12 can go
anywhere

Direct Mapped

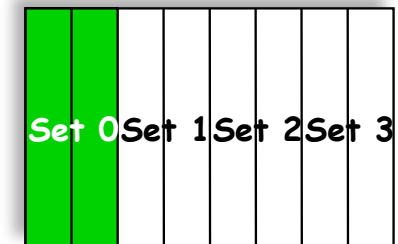
1 2 3 4 5 6 7 8



Block 12 can go
only into Block 4
($12 \bmod 8$)

Set Associative

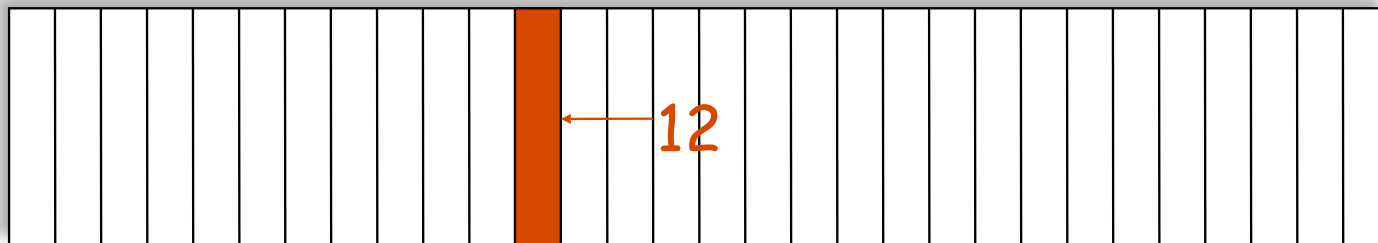
1 2 3 4 5 6 7 8



Block 12 can go
anywhere in set 0
($12 \bmod 4$)

Memory:

1 2 3 4 5 6 7 8 9.....



Memory access equations

- Using what we defined on previous slide, we can say:
 - **Memory stall clock cycles =**
 - Reads x Read miss rate x Read miss penalty +
 - Writes x Write miss rate x Write miss penalty
- Often, reads and writes are combined/averaged:
 - **Memory stall cycles =**
 - Memory access x Miss rate x Miss penalty (approximation)
- Also possible to factor in instruction count to get a “complete” formula:

$$CPU\ time = IC \times \left(CPI_{execution} + \frac{Memory\ stall\ clock\ cycles}{Instruction} \right) \times Clock\ cycle\ time$$

Reducing cache misses

- Obviously, we want data accesses to result in cache hits, not misses –this will optimize performance
- Start by looking at ways to increase % of hits....
- ...but first look at 3 kinds of misses! **More devices = more cache to help reduce**
 - **Compulsory misses:**
 - Very 1st access to cache block will not be a hit –the data's not there yet!
 - **Capacity misses:**
 - Cache is only so big. Won't be able to store every block accessed in a program – must swap out!
 - **Conflict misses:**
 - Result from set-associative or direct mapped caches
 - Blocks discarded/retrieved if too many map to a location