10-dpath-notes

The Datapath and the ISA

Agenda

(Textbook: 4.0 through 4.1.3, and Fig 4.22)

- 1. Doing Some Work
- 2. Key Concept: Hide the details
- 3. Key concept: the program
- 4. The Datapath
- 5. The Instruction Set Architecture
- 6. Addressing Modes

---

0. Re-Orienting

Electricity

Transistors

Gates

Combinatorial logic

Feedback to create memory

Sequential logic

Hanging components off common buses, and moving stuff around

---

1. Doing Some Work

Let's continue with the datapath from Monday

Think about how we might do things like:

- "Move a word from address X to register Y"

How would we wiggle the control lines to do it?

How could we make that automatic?

What about things like...

- "Move a word from register Y to address X"
- "Move a word from register Y to register Z"
- "Add register Y and register Z, and store them in register A"
- "Use the contents of register Y as an address, and move the word there to register A"
- "Add the contents of register Y to this address, and move the word at that address to register Z"

---

2. Key Concept: Hide the details

Abstracting

(draw out on board)

memory

- and to make life easier.... maybe a **memory address register**
- and a **memory data register**
- or even two MDRs: for writing to memory, and reading from it
- and heck.... what if memory is slower? MemReq, MemReady

bus

- and distinguish between outside the processor (slower) and insider (faster)

registers

control lines

Think again about things like

Loading a register from memory

- specifying all those valves to wiggle
- and maybe many steps of that
- hey, microcode! maybe driven by an FSM!

Do you really want to worry about how to load a register?

So... abstract the "please do this sequence of steps of microcode" to an **instruction**

- coded as some binary value

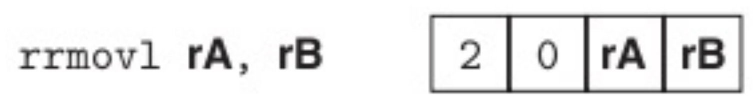
3. Key concept: the program

a sequence of instructions

oops, better need a "PC" or "IP"

method of fetching instructions

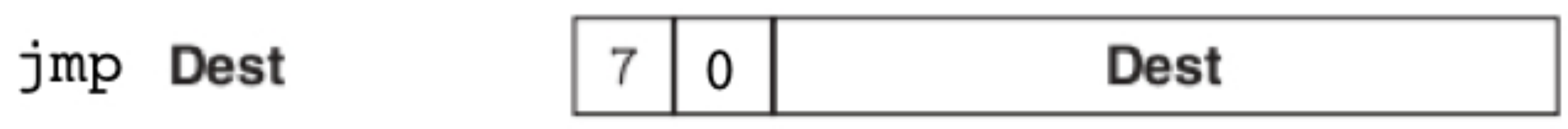
then figuring out what each one does, and wiggling the control lines appropriately



(From Fig 4.2)

There are competing conventions... but let's say "src" comes before "dst"

Or this variation from 4.2:



- "Jump" to that "dest" value
- what does that mean?
- how do we actually DO that?)

Where should the program reside?

"Obviously" in main memory (e.g. the address space). The **Von Neumann** model.

Q: how does the computer find a program when you first turn it on?

But this wasn't always obvious---e.g., the **Harvard architecture**.

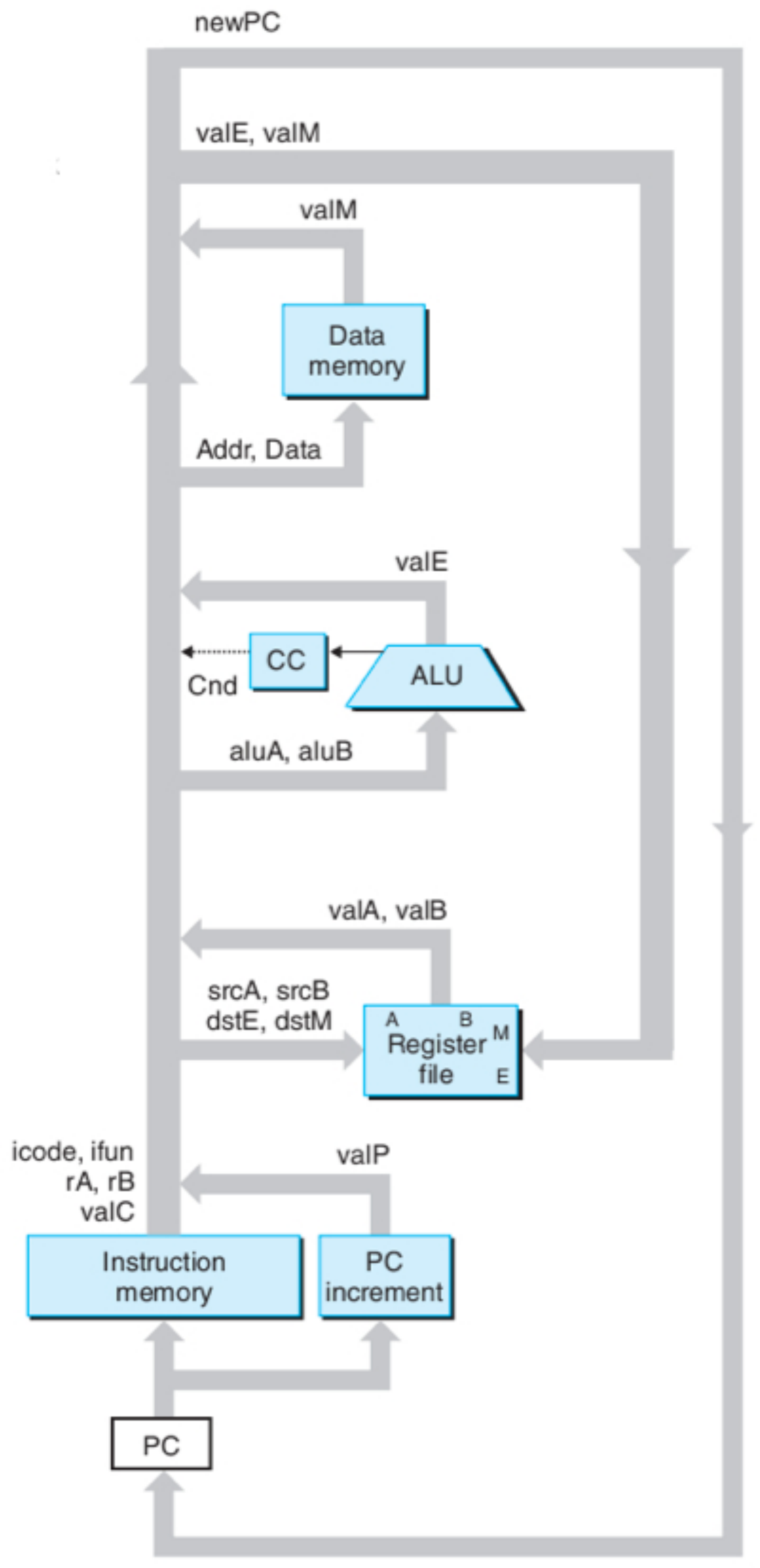
http://www.harvardclubofjapan.org/public_img/events/K06_JohnHarvard2.jpg

4. The Datapath

Other pieces..

- an ALU (arithmetic logic unit)
- state flags, aka condition codes

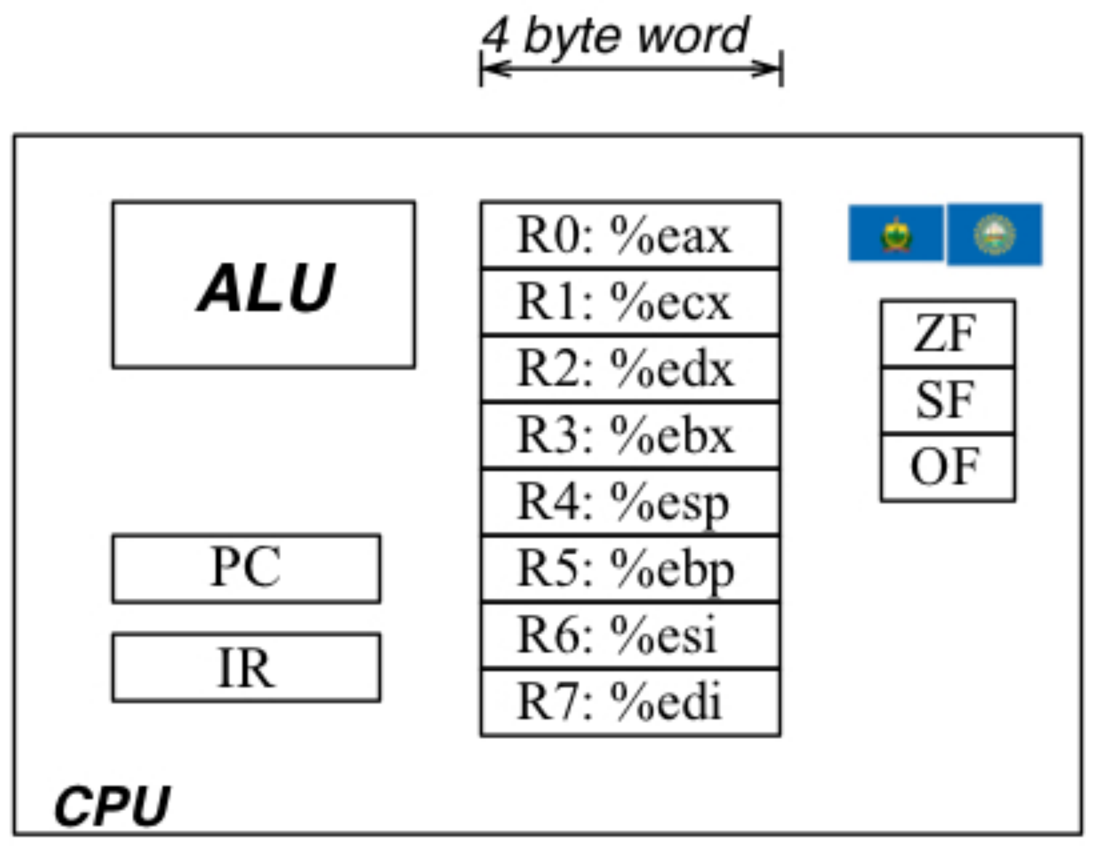
Putting it altogether... Fig 4.22!

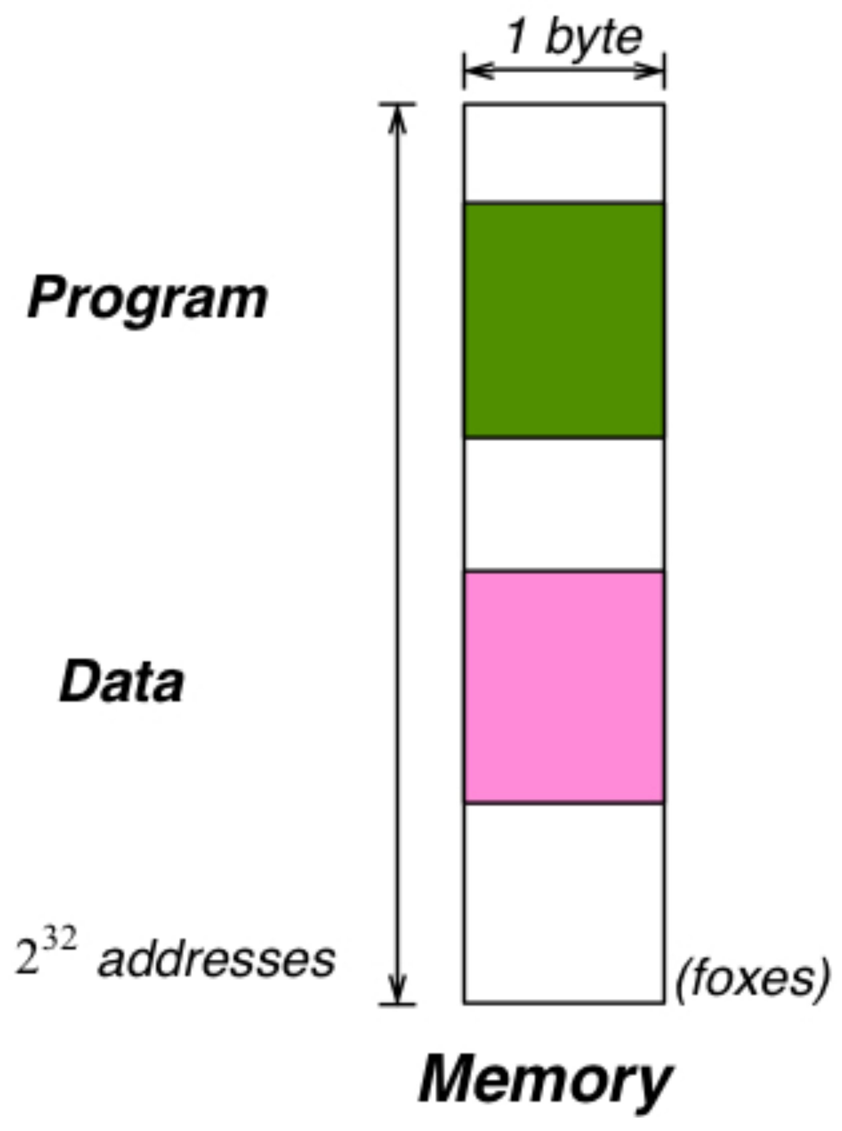


5. The Instruction Set Architecture

Stuff to Play With

In the Y86:





Word size

two's complement

Address space

Registers (general purpose ones, special ones...and some in-between)

PC....

ALU

Condition codes

ZF

SF (e.g. "negative"

OF

changed after each ALU operation)

(Real-world CPUs have some additional complexity, btw)

Ways to Play With It

In the Y86:

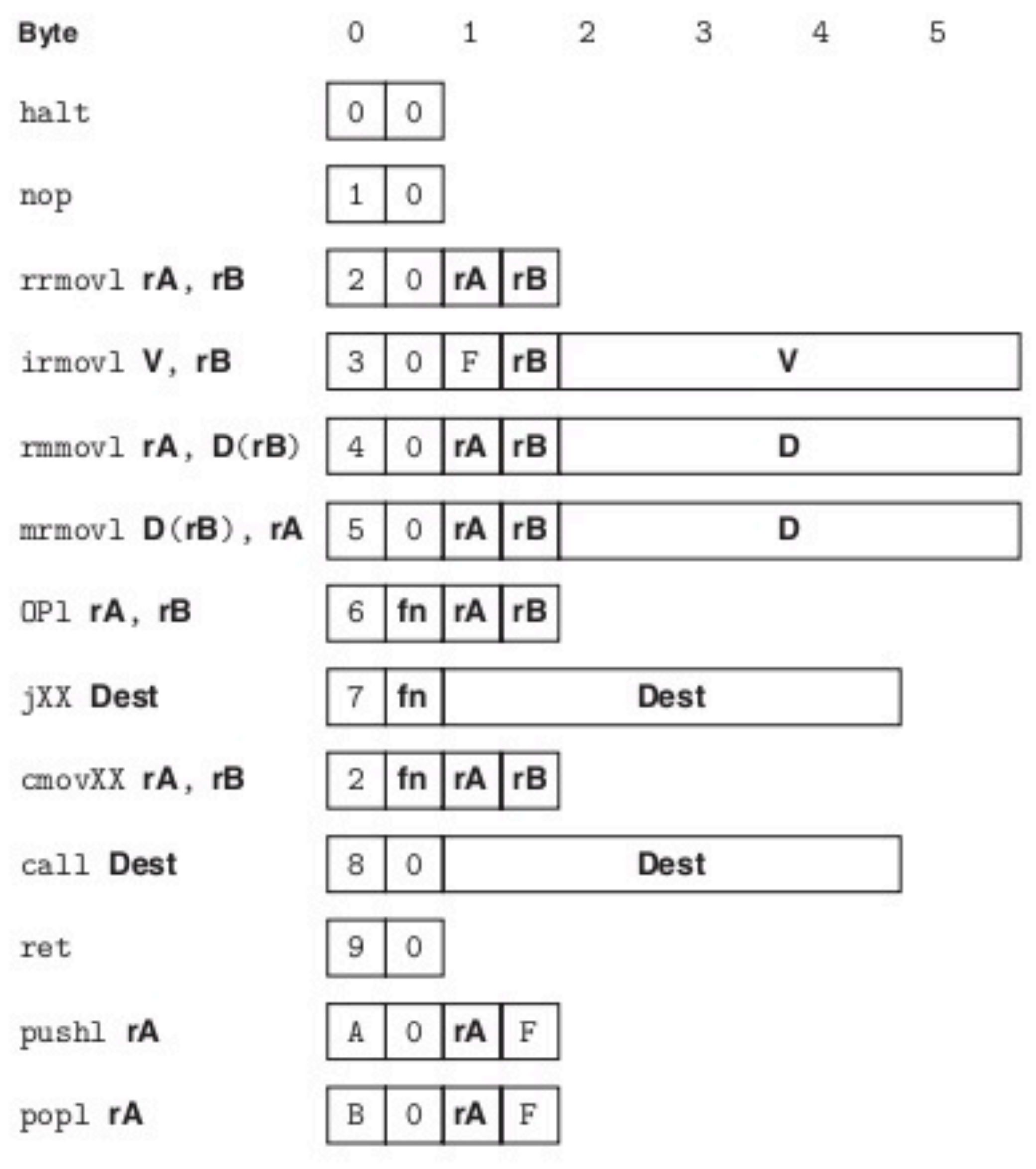


Figure 4.2
Y86 instruction set. Instruction encodings range between 1 and 6 bytes. An instruction consists of a 1-byte instruction specifier, possibly a 1-byte register specifier, and possibly a 4-byte constant word. Field `fn` specifies a particular integer operation (`OPl`), data movement condition (`cmovXX`), or branch condition (`jXX`). All numeric values are shown in hexadecimal.

Some textbooks like to partition instructions into three kinds:

- "operate"
- "move data"
- "control"

---

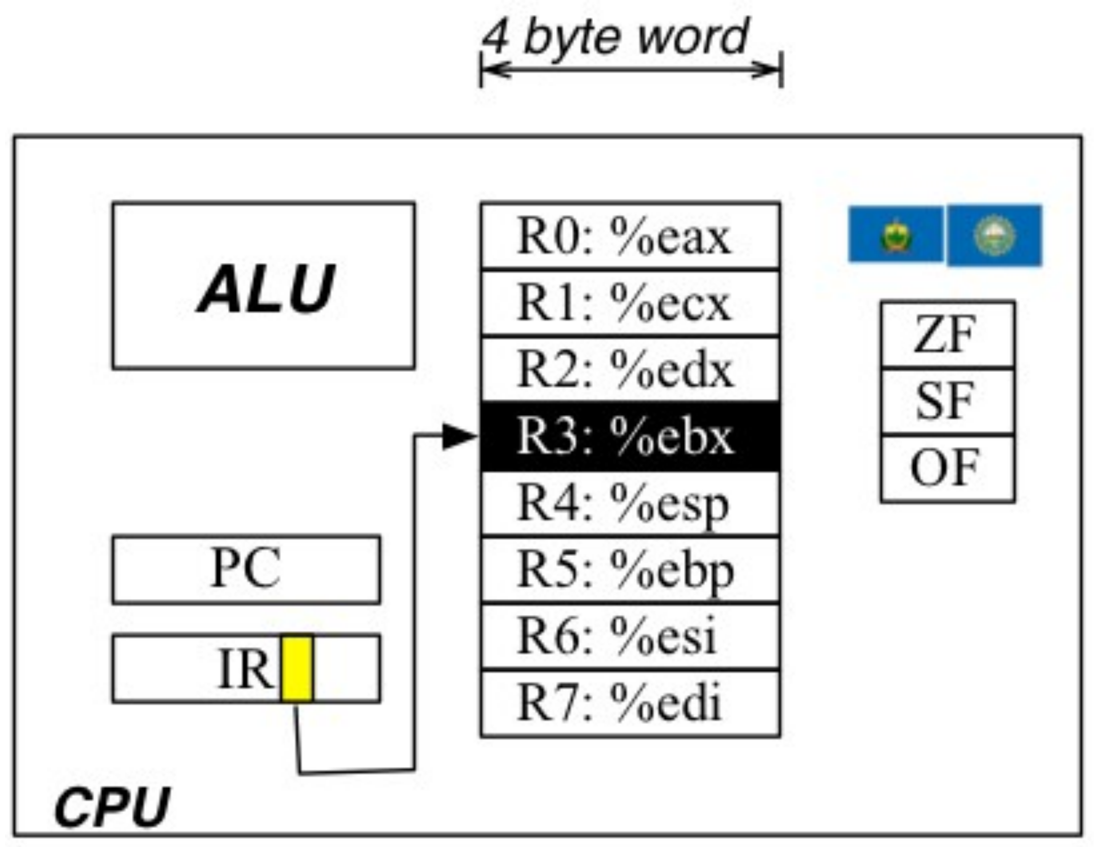
6. Addressing Modes (which we did not get to)

How does an opcode specify the data?

In the real world, there are many variations, and some are hard to distinguish.

Here are the principal ones in the Y86.

Register



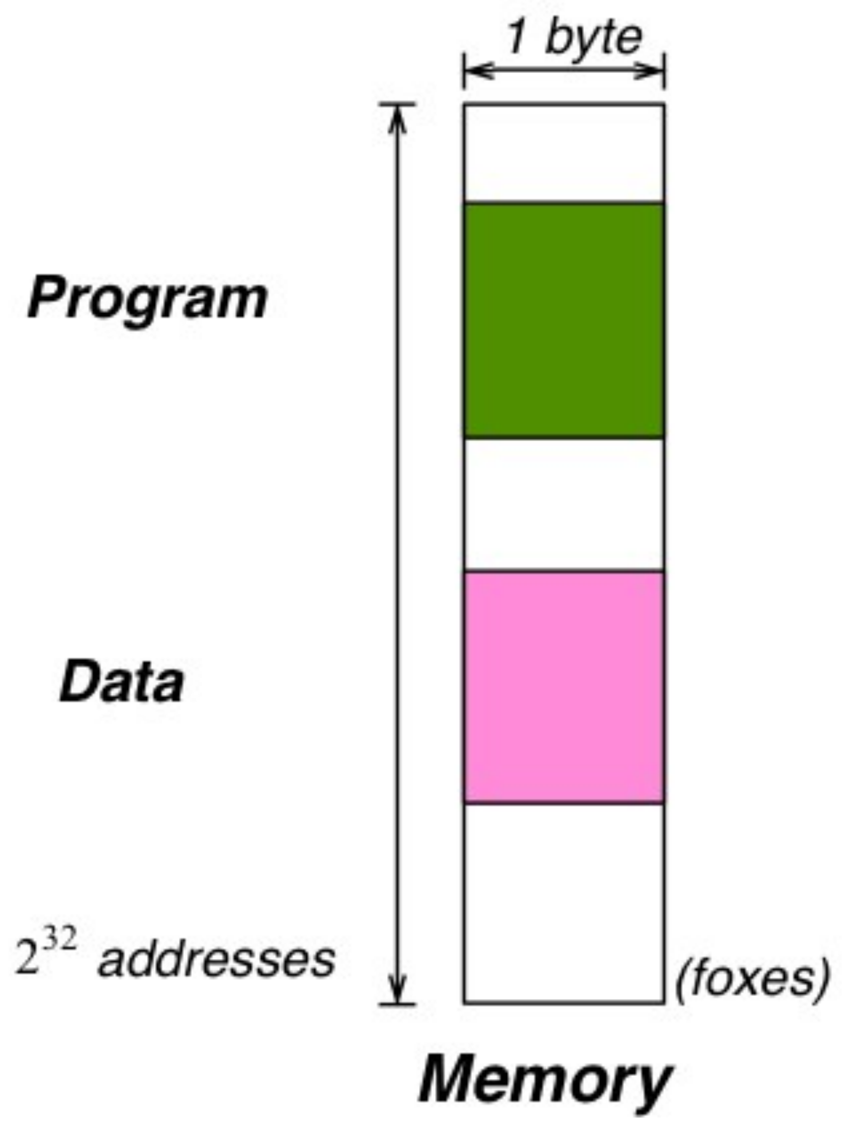
operand

reg index

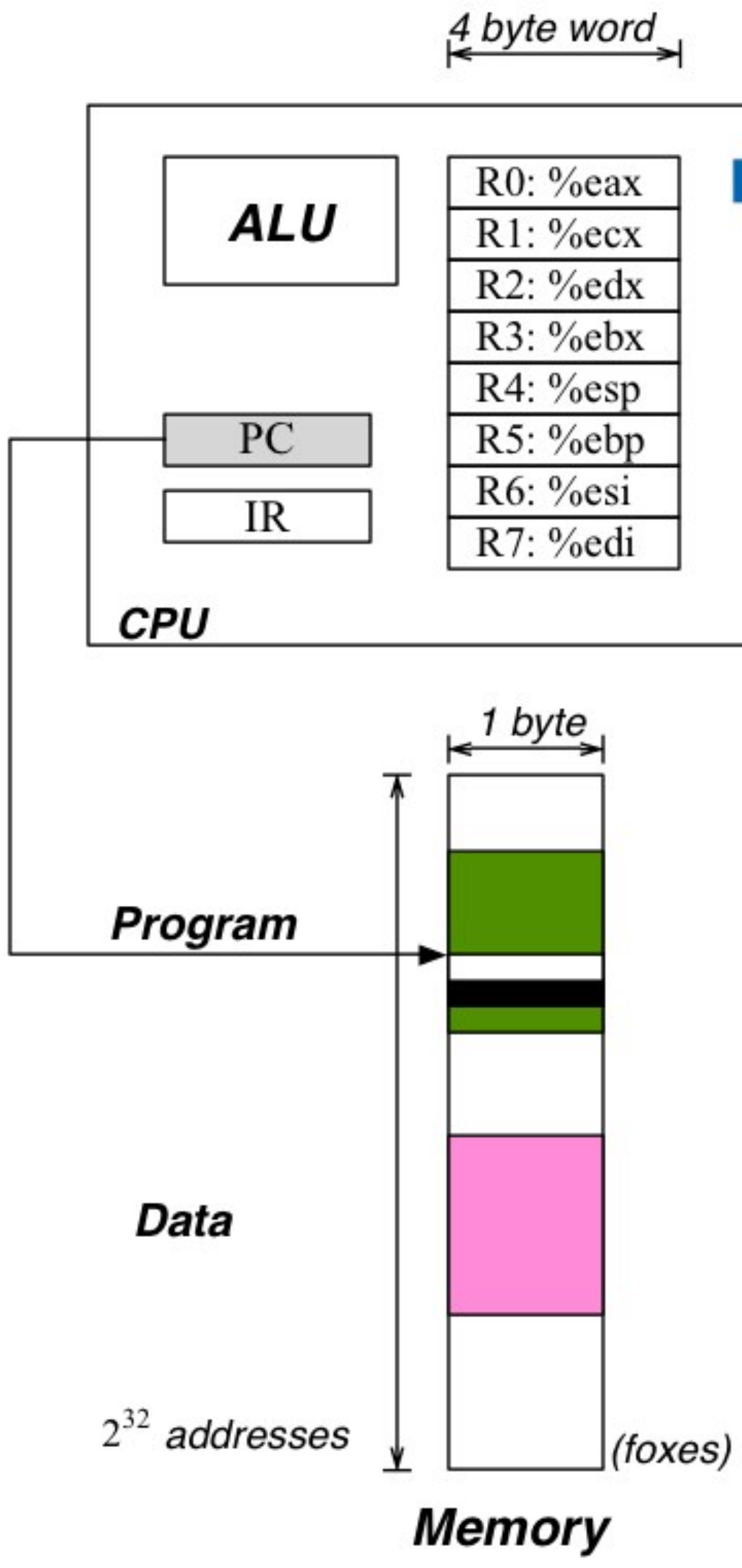
base

offset

address of data



Immediate



operand

reg index

base

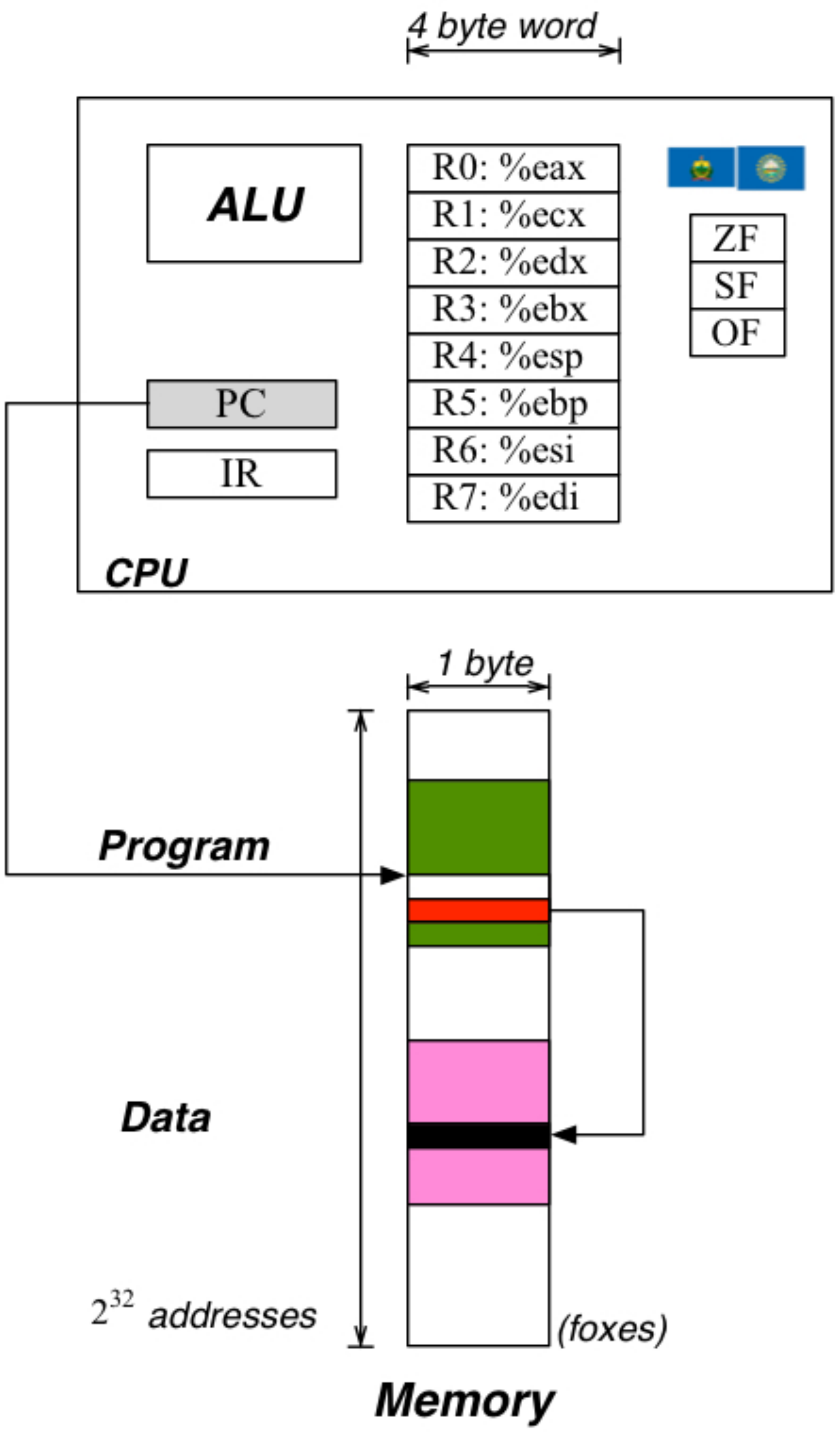
offset

address of data

Absolute

ABSOLUTE

(a better example :)



operand

reg index

base

offset

address of data

`rmmovl` **rA, D**

4	0	rA	F	D

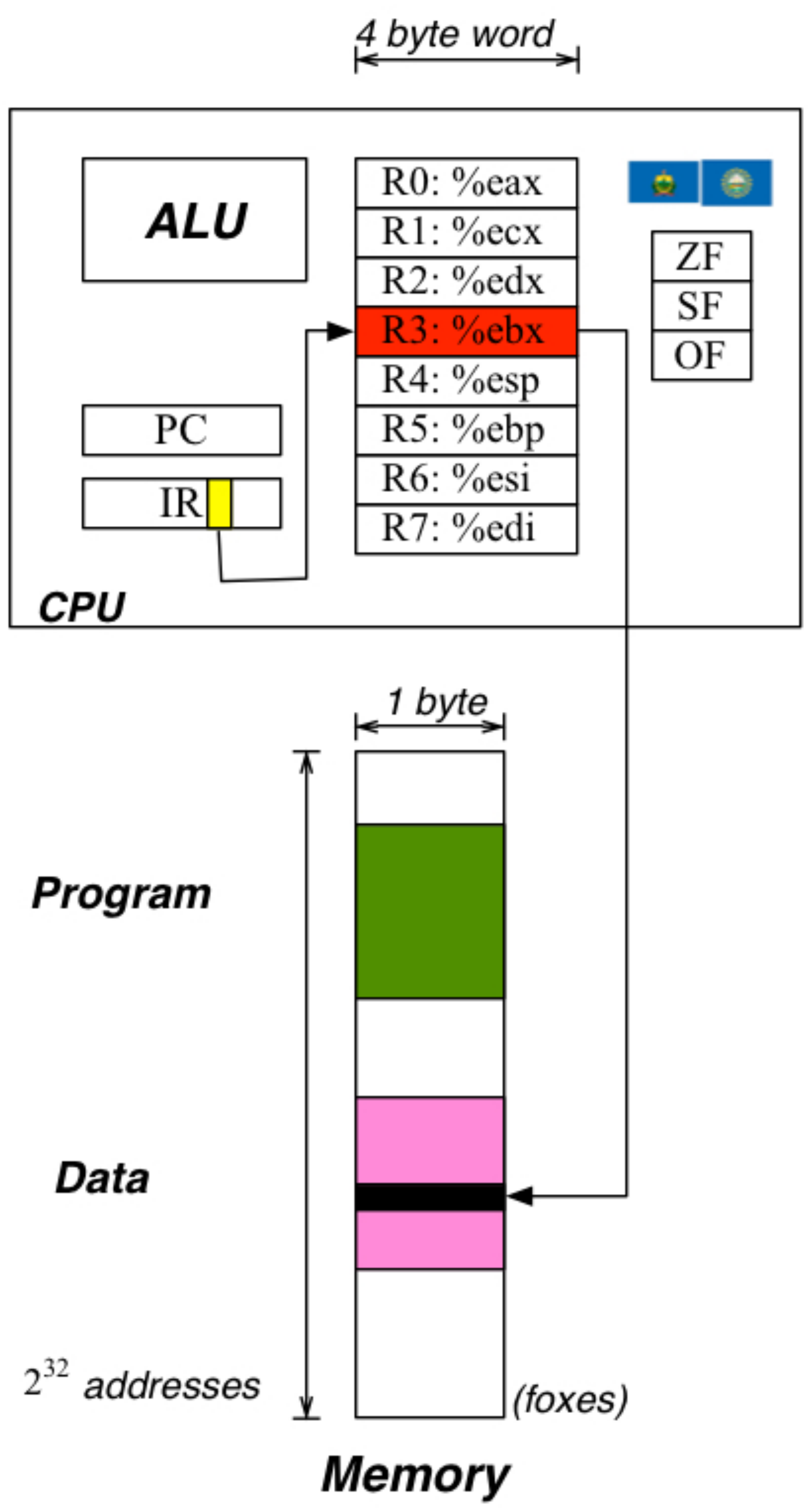
(Fig 4.2 fragment)

(move rA to the address D)

Indirect

INDIRECT

(fixed :)



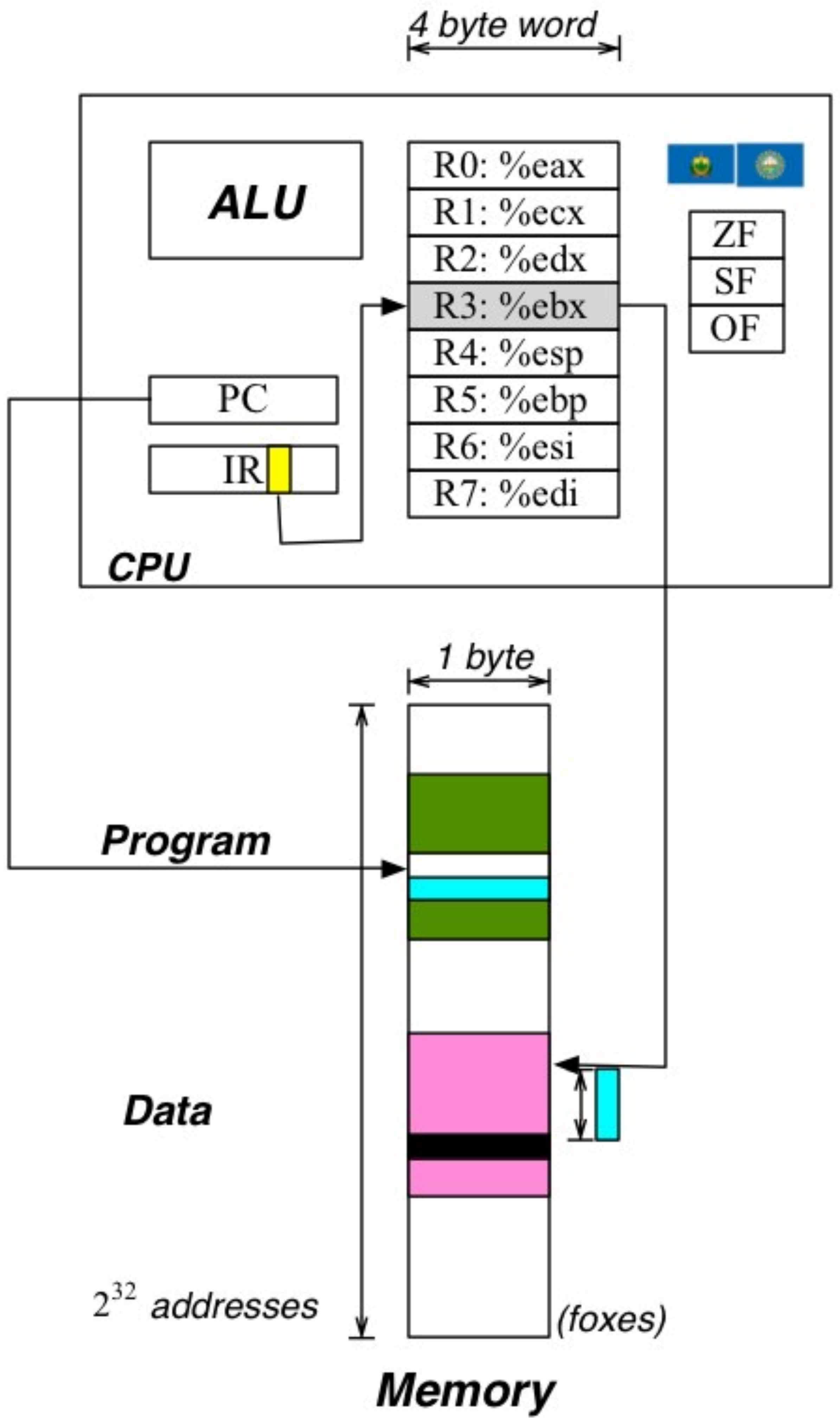
- operand
- reg index
- base
- offset
- address of data

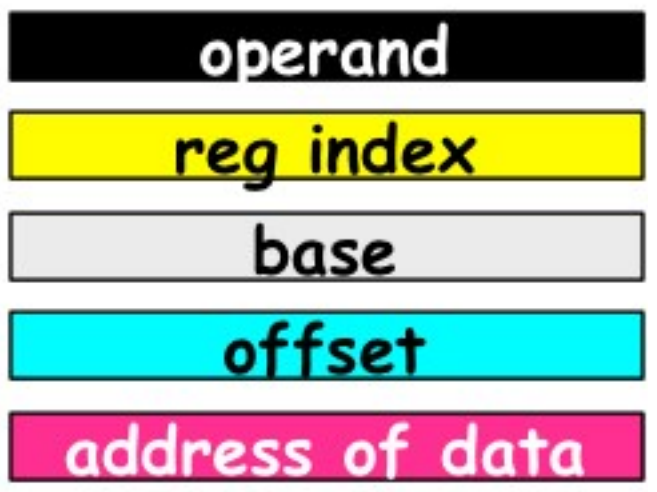
`rmmovl` **rA**, (**rB**)	4	0	**rA**	**rB**	00 00 00 00

(Fig 4.2 fragment)

Move rA to the address in rB

Base + Displacement


4 byte word
ALU
R0: %eax
R1: %ecx
R2: %edx
R3: %ebx
R4: %esp
R5: %ebp
R6: %esi
R7: %edi
ZF
SF
OF
PC
IR
CPU
1 byte
Program
Data
2^{32} addresses
(foxes)
Memory


operand
reg index
base
offset
address of data