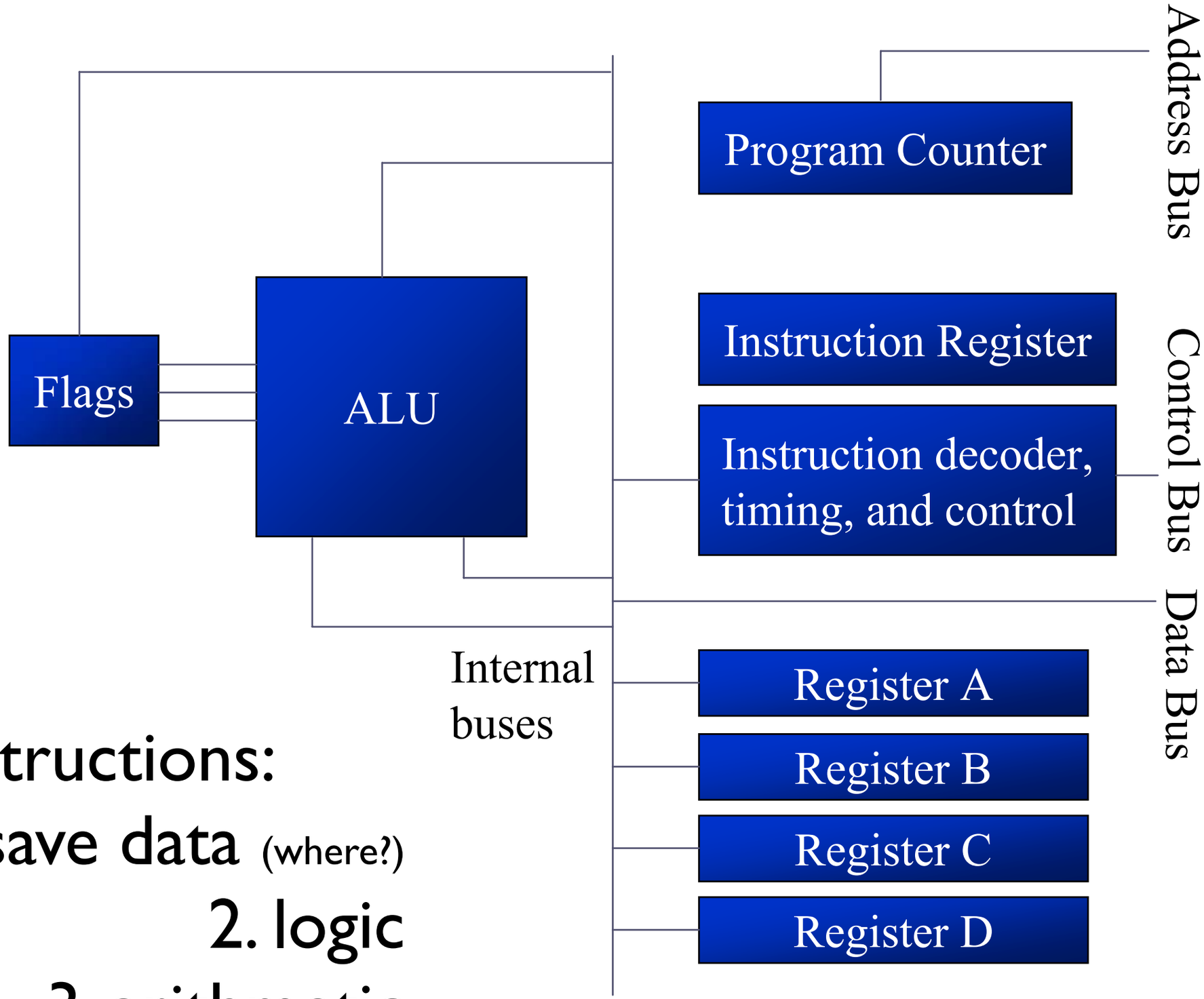
Architecture II

ECE 3710

Not only do I not know what's going on, I wouldn't know what to do about it if I did.

- George Carlin

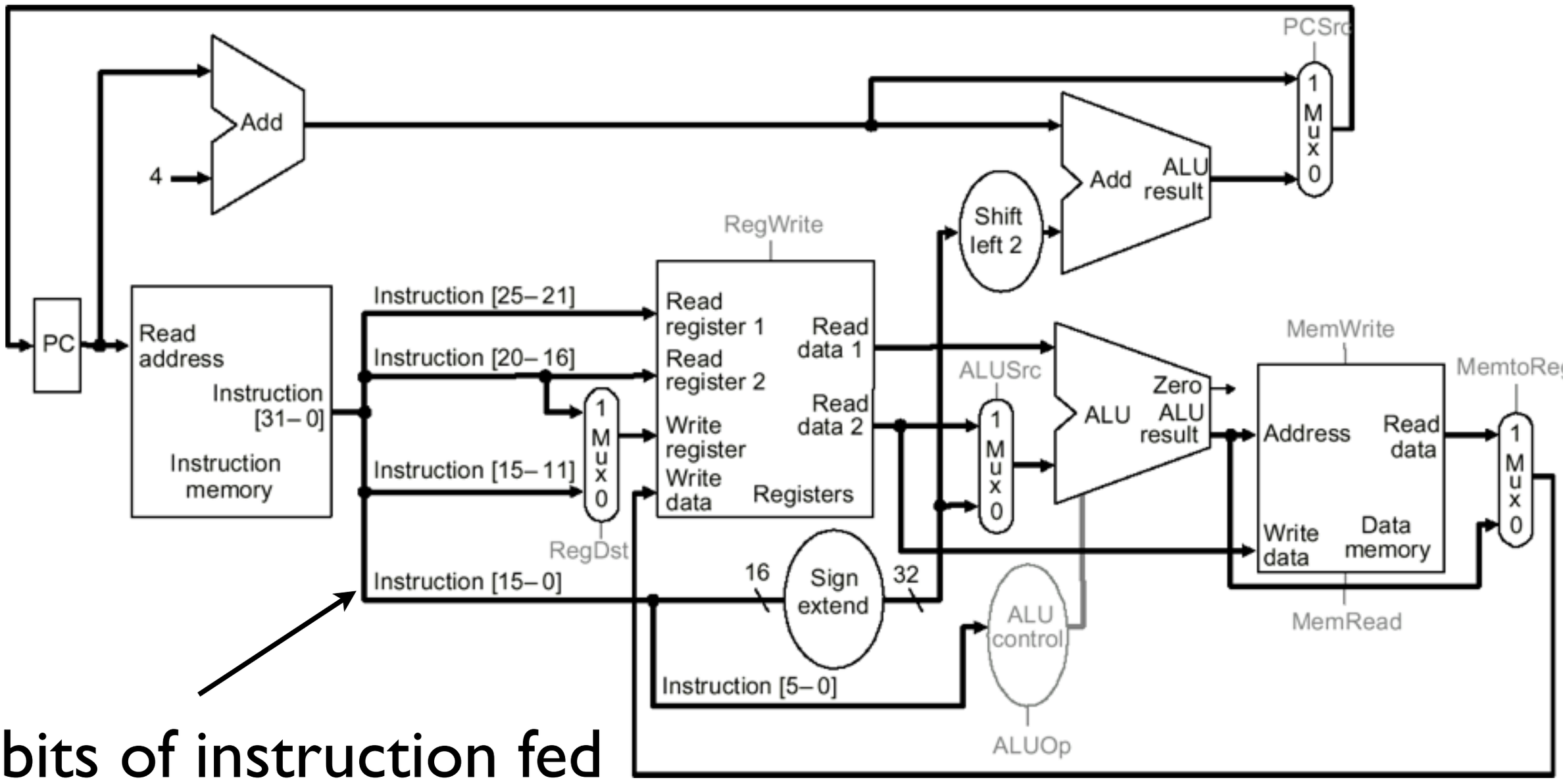
how do we use this?



computer instructions:

1. load/save data (where?)
2. logic
3. arithmetic

32-bit single cycle MIPS



bits of instruction fed to various components

8/16-bit example

program we write:

Action	Code	Data
Move value 21H into register A	B0H	21H
Add value 42H to register A	04H	42H
Add value 12H to register A	04H	12H

a single instruction (0x0412)

these bits turn certain components on and off

8/16-bit example

what it looks like in memory:

points to byte addr.

PC

(increases by two [16-bit alignment])

Memory address	Contents of memory address
1400	(B0) code for moving a value to register A
1401	(21) value to bc moved
1402	(C4) code for adding a value to register A
1403	(42) value to be added
1404	(04) code for adding a value to register A
1405	(12) value to be added
1406	(F4) code for halt

a single instruction

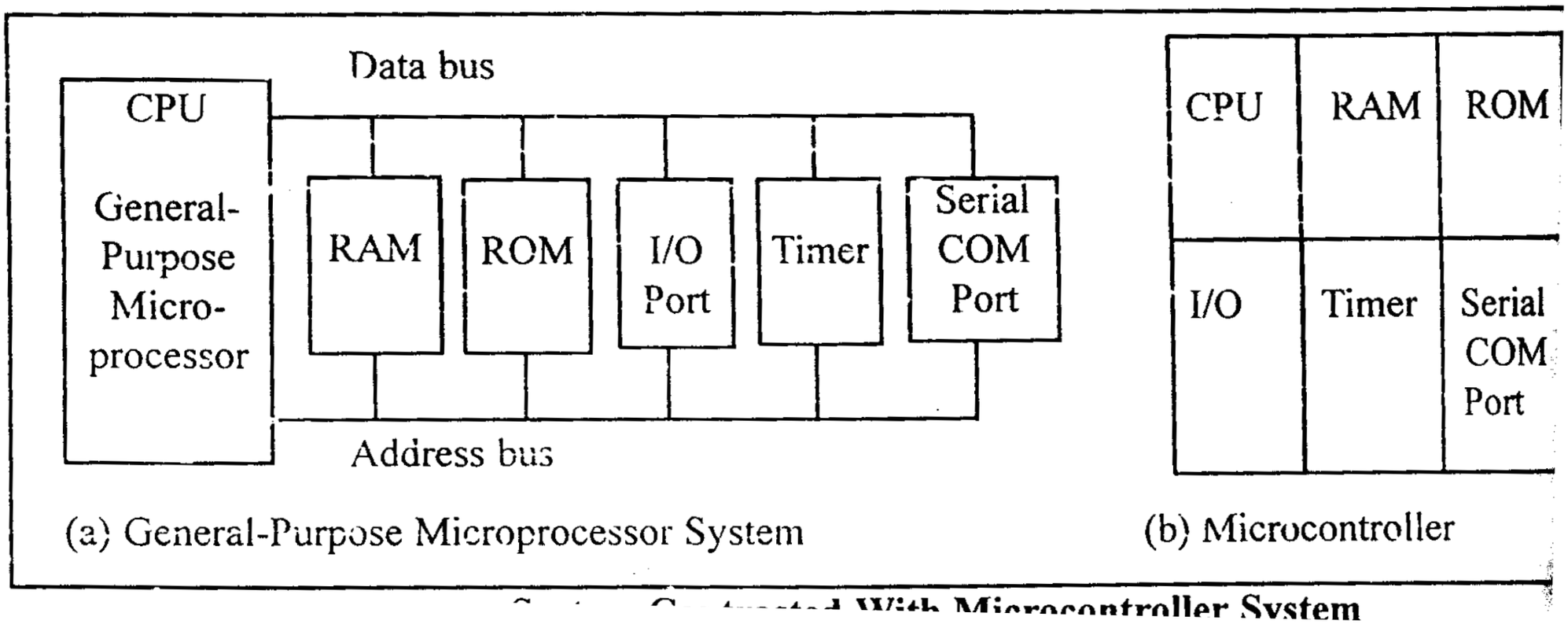
notice the size...bytes...an 8/16-bit CPU

(painful)

tell someone you have an 8/16-bit CPU...

which is why we use a 32-bit one...more on this later

diversion: μProcessor vs. μController



Thursday, September 5, 13

uController < uProcessor:

1. cost
2. computation
3. memory
4. power
5. uses (?)

which uController?

the non-engineering response to tech decisions...

uController considerations:

1. speed
2. package
3. power
4. memory
5. I/O
6. upgrade
7. $/unit (discounts)
8. IDE
9. availability

distinguishing characteristics

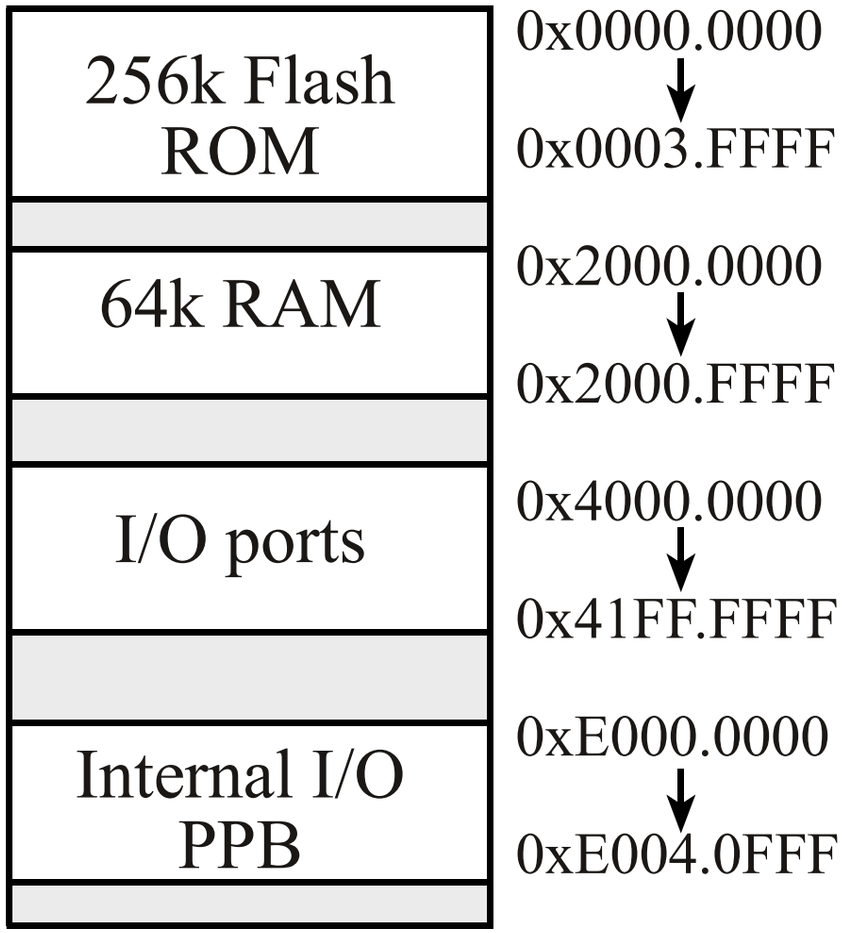
Part number	*RAM*	*Flash*	*I/O*	*I/O modules*
LM3S811	8	64	32	Timer, ADC, PWM, serial
LM3S1968	64	256	52	Timer, ADC, PWM, serial
LM3S2965	64	256	56	Timer, ADC, PWM, serial, CAN
LM3S3748	64	128	61	Timer, ADC, PWM, serial, DMA, USB
LM3S6965	64	256	42	Timer, ADC, PWM, serial, Ethernet
LM3S8962	64	256	42	Timer, ADC, PWM, serial, CAN, Ethernet, IEEE1588
LM3S9B90	96	256	60	Timer, ADC, serial, CAN, Ethernet, USB
LM3S9B92	96	256	65	Timer, ADC, PWM, serial, CAN, Ethernet, USB
	KiB	KiB	pins	

back to architecture

memory map of ARM uC:

your program goes here

it puts stuff here

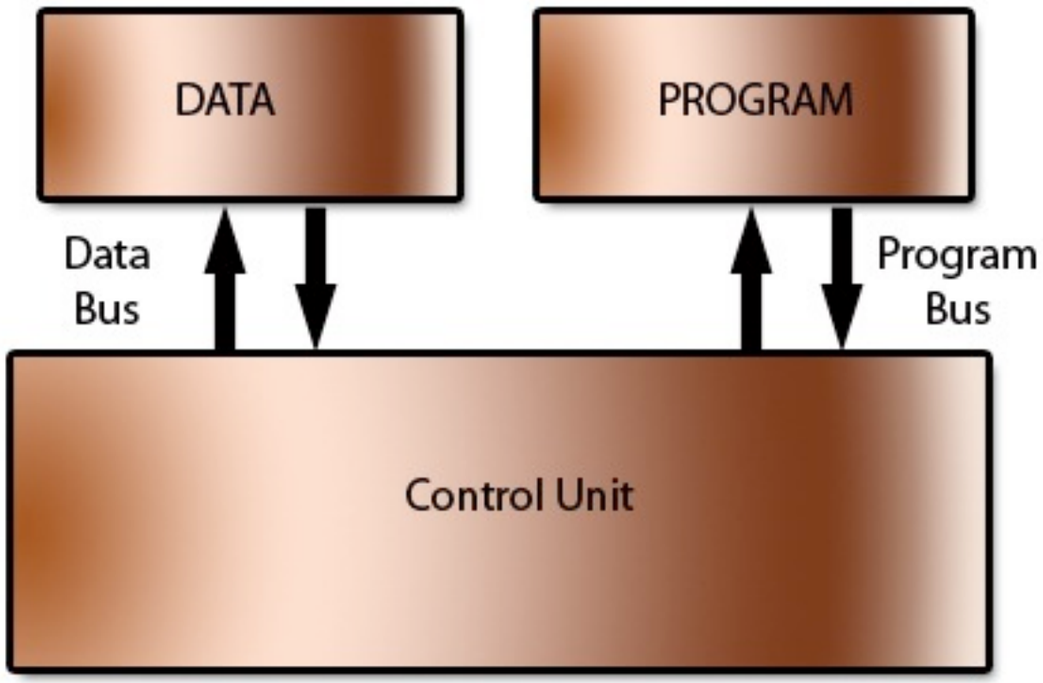


memory mapped I/O... mmmm...

this division not typical...

addressing data/instructions

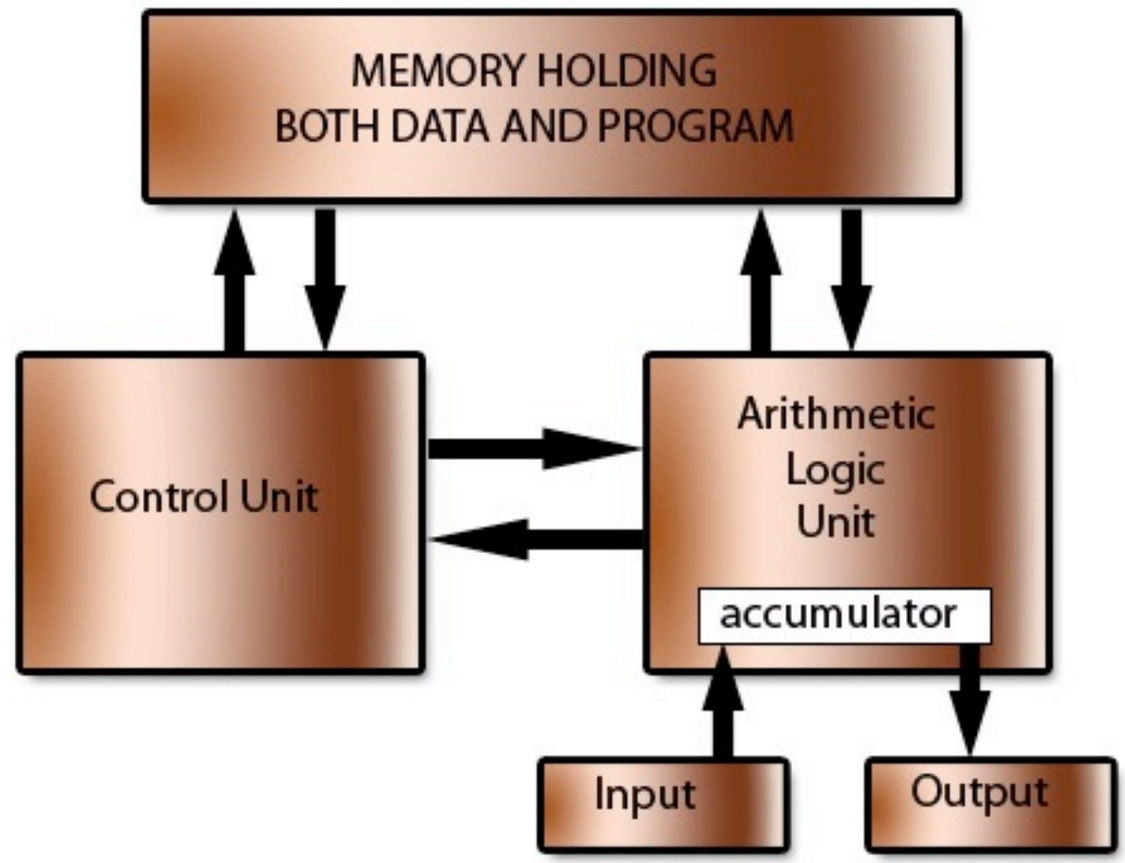
The Harvard architecture



(c) www.teach-ict.com

separate

The Von Neumann or Stored Program architecture



(c) www.teach-ict.com

unified

addressing data/instructions

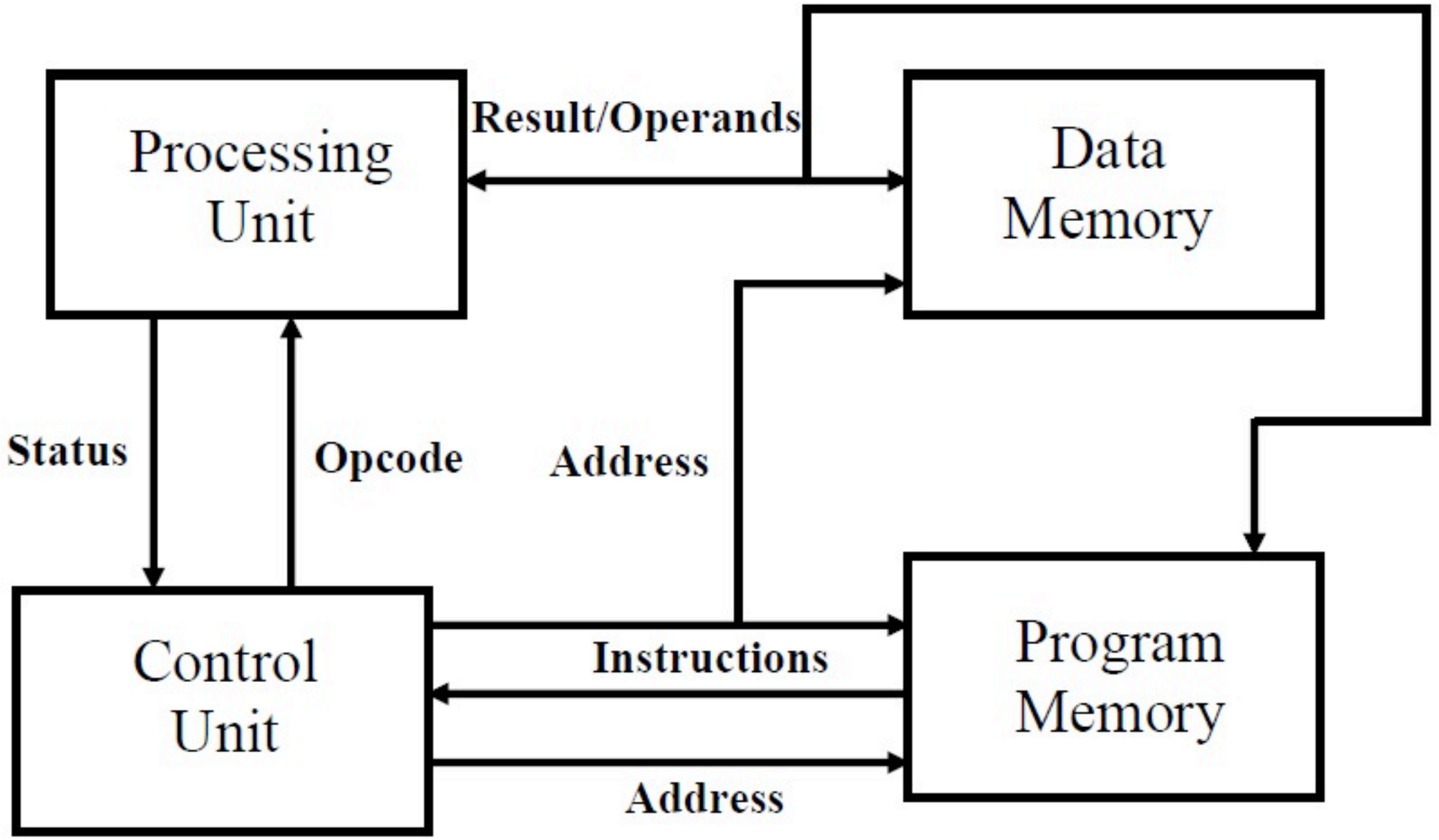
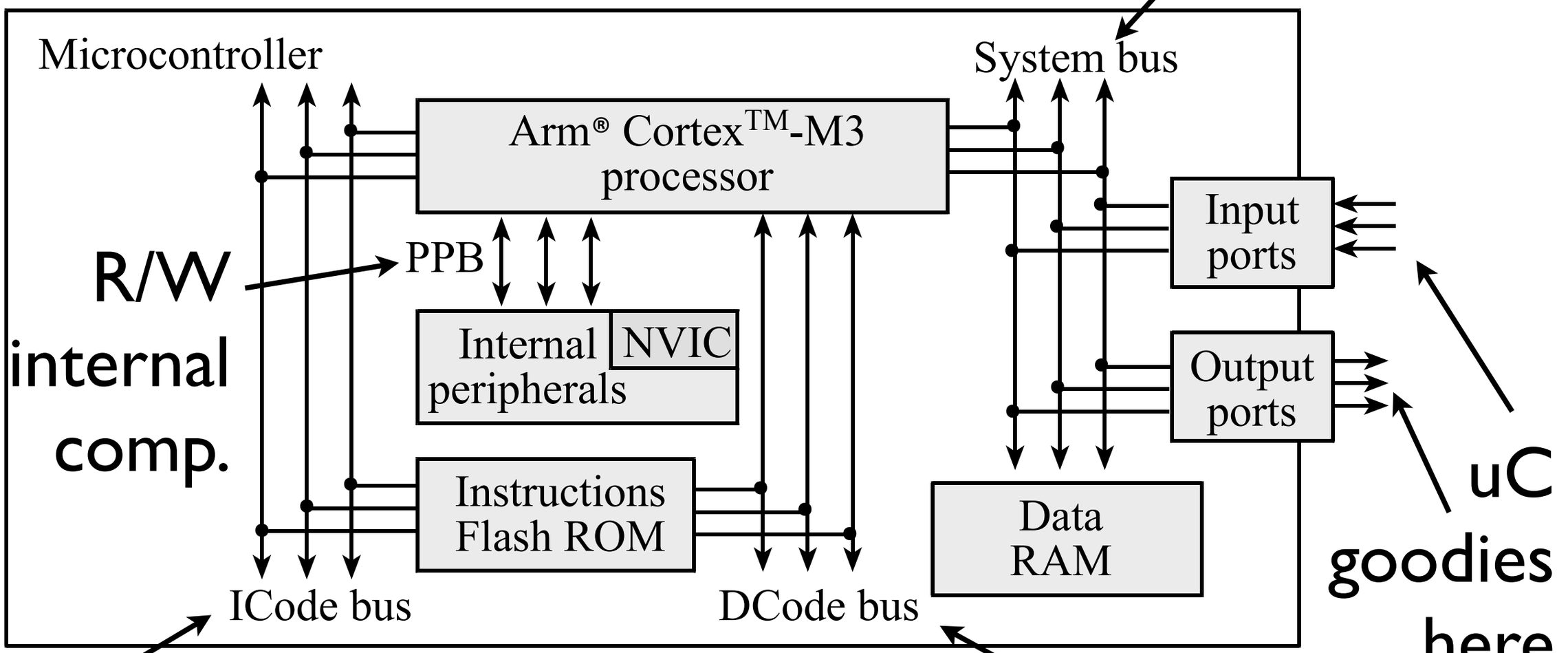


Fig. 4.3 The modified Harvard architecture

separate but seen as unified

(can access program mem as data mem)

R/W external data/ peripherals; instructions



R/W internal comp.

uC goodies here

fetch opcode (instruction)

ARM Cortex-M3 Harvard Architecture

modified

R const. data (fetch instructions and data at same time)

32-bit example of program execution

Actual ARM® Cortex™-M3 processor	Simplified processor
Sometimes 8-, 16-, 32-bit access	All opcode accesses are aligned 16-bit
Special case for unaligned access	All data accesses are aligned 32-bit
Instruction queue enhances speed	Simple fetch-execute sequence
Fetches op codes for later execution	Fetches op codes for immediate execution
Fetches op codes that are never executed	Fetched op codes are always executed
Five buses with simultaneous accessing	One shared bus
Harvard architecture	Von Neumann architecture

characteristics of modern processor

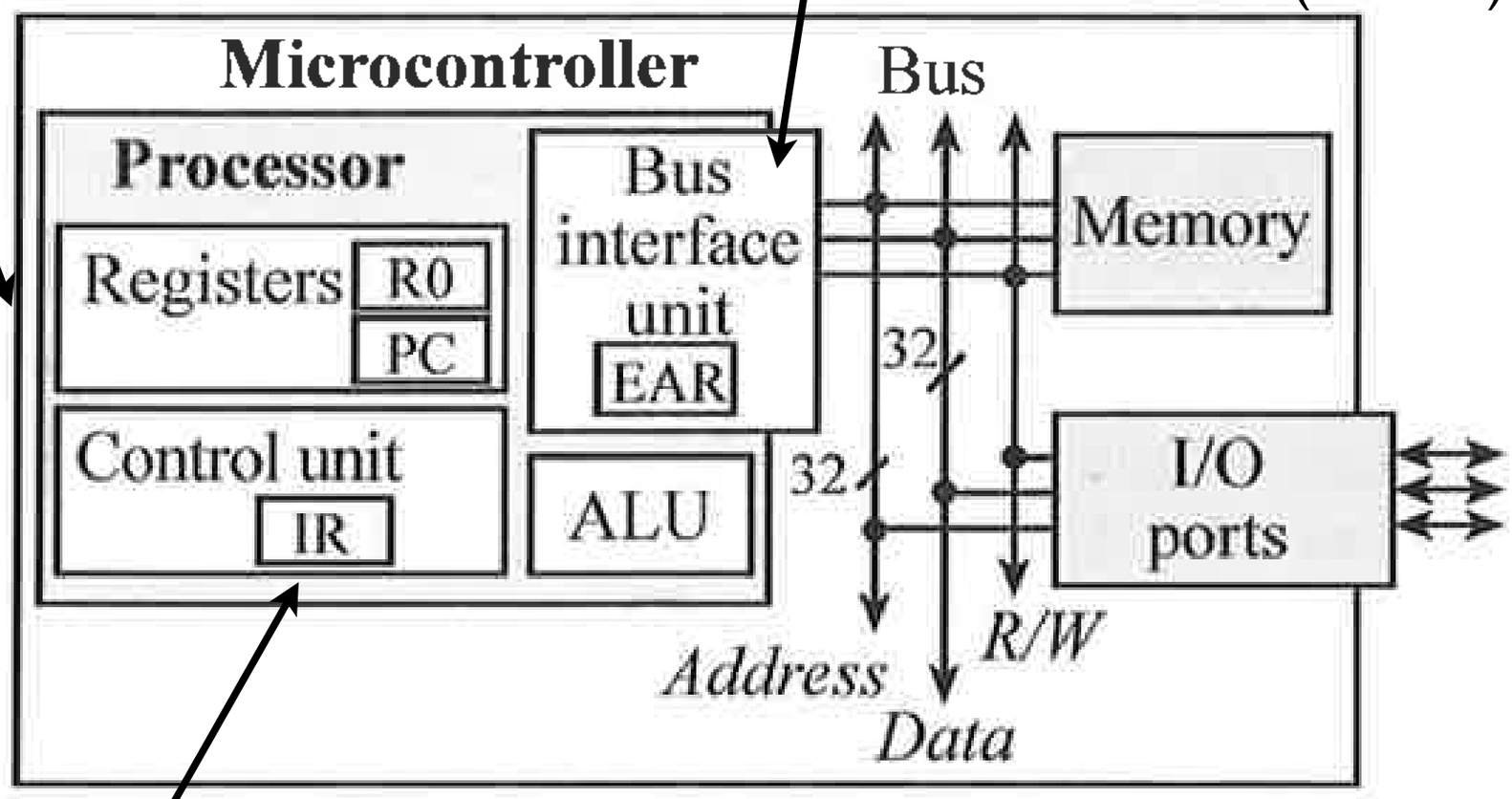
(except Harvard)

PC: points to next instruction to execute

effective address register

EAR: src/dst address for operand

(indirect)



IR: instruction being executed

R: instr; data (EAR); stack

W: data (EAR); stack