Architecture III

ECE 3710

May the forces of evil become confused on the way to your house.

- George Carlin

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

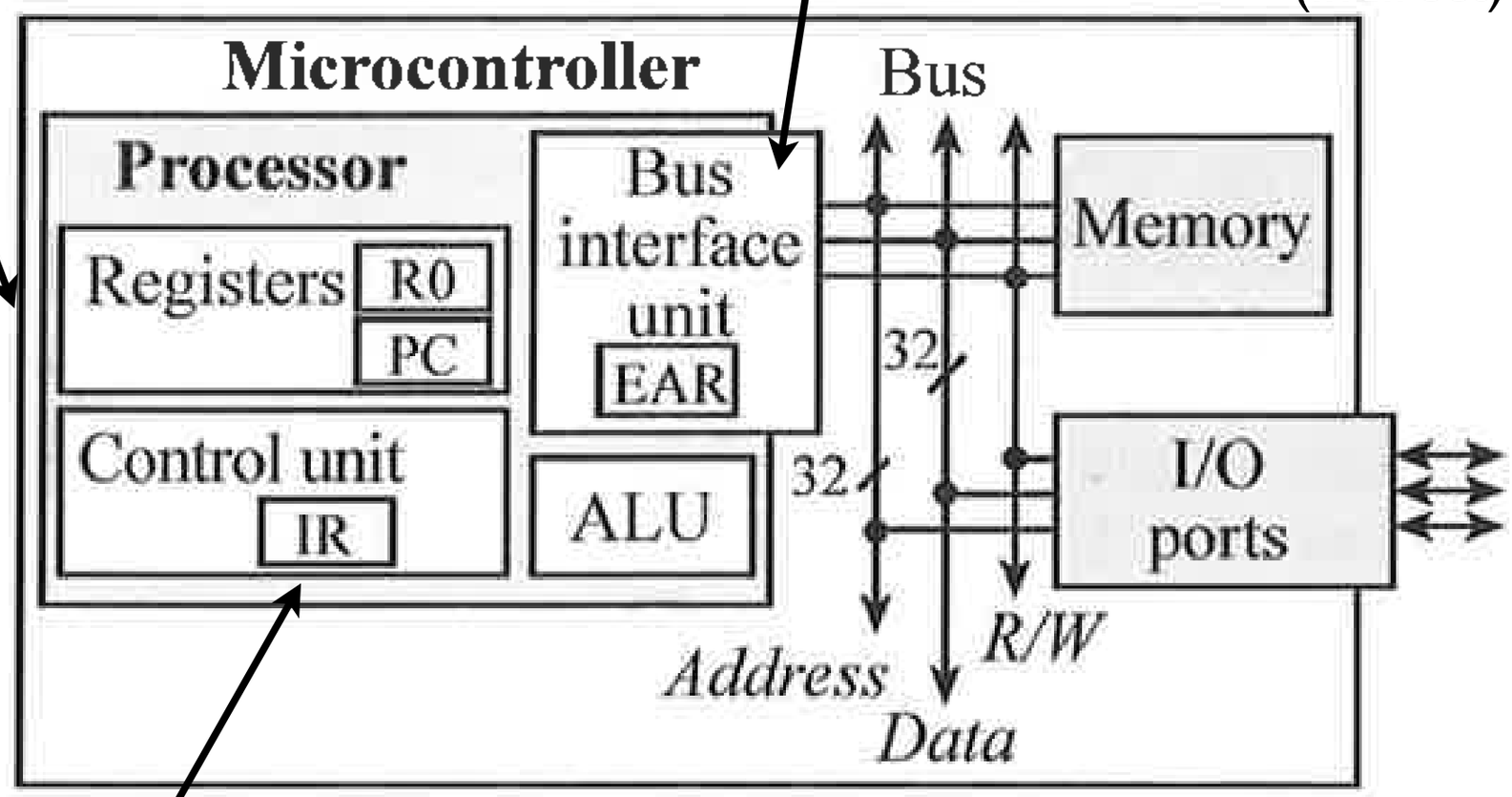
characteristics of modern processor

(except Harvard)

PC: points to next instruction to execute

effective address register

EAR: src/dst address for operand (indirect)

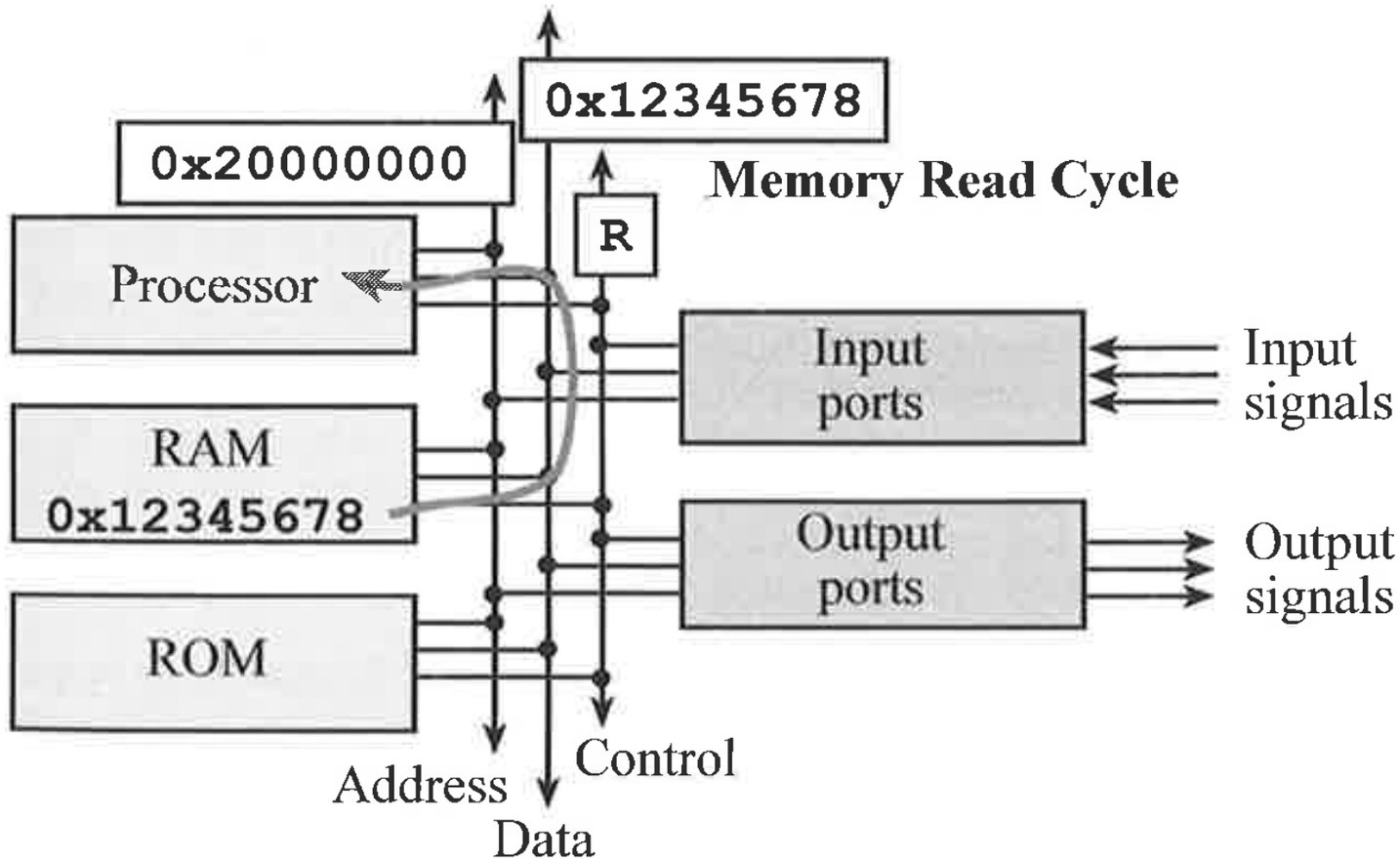


IR: instruction being executed

R: instr; data (EAR); stack

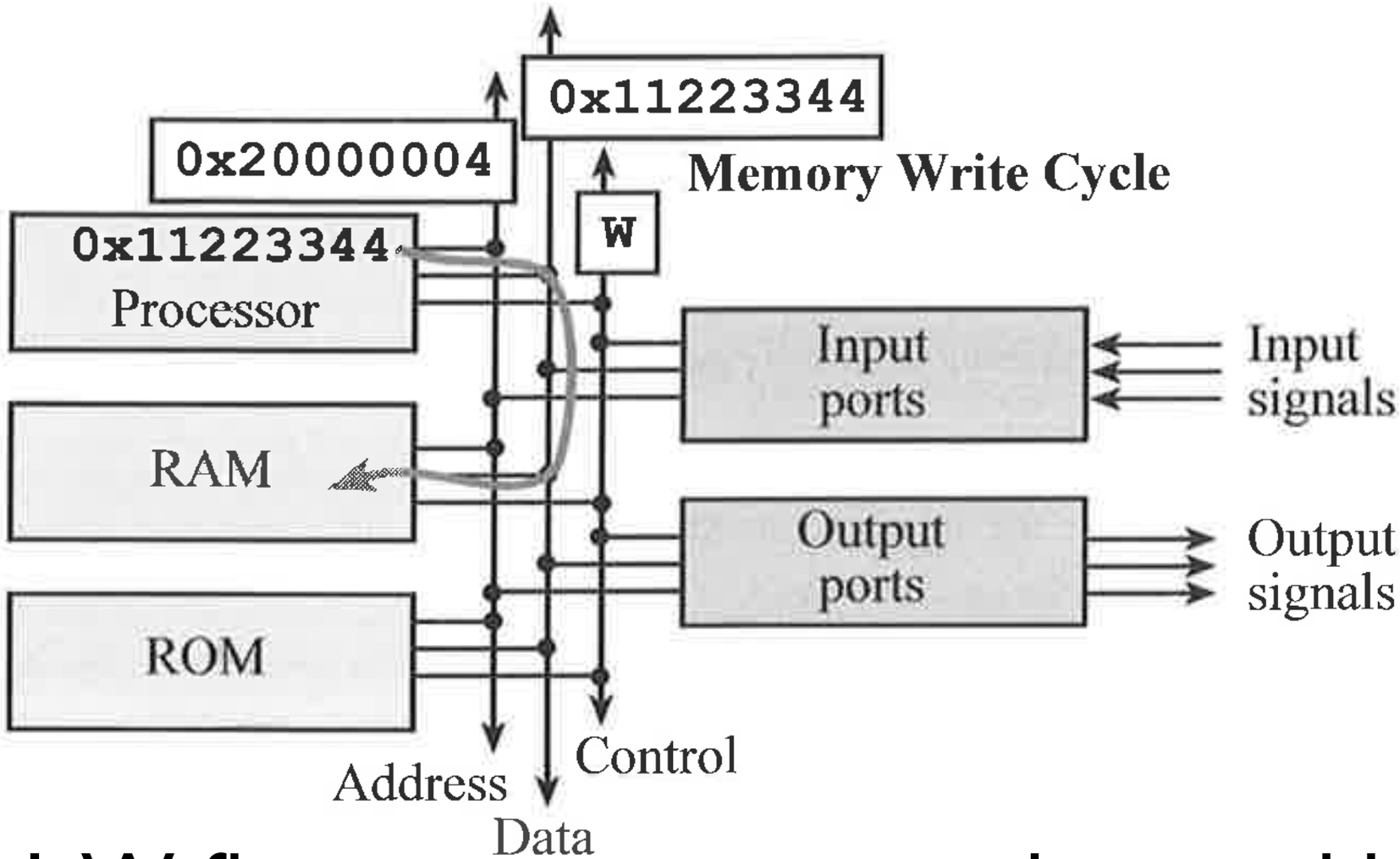
W: data (EAR); stack

R: addr. 0x2000.0000 contains 0x123456789



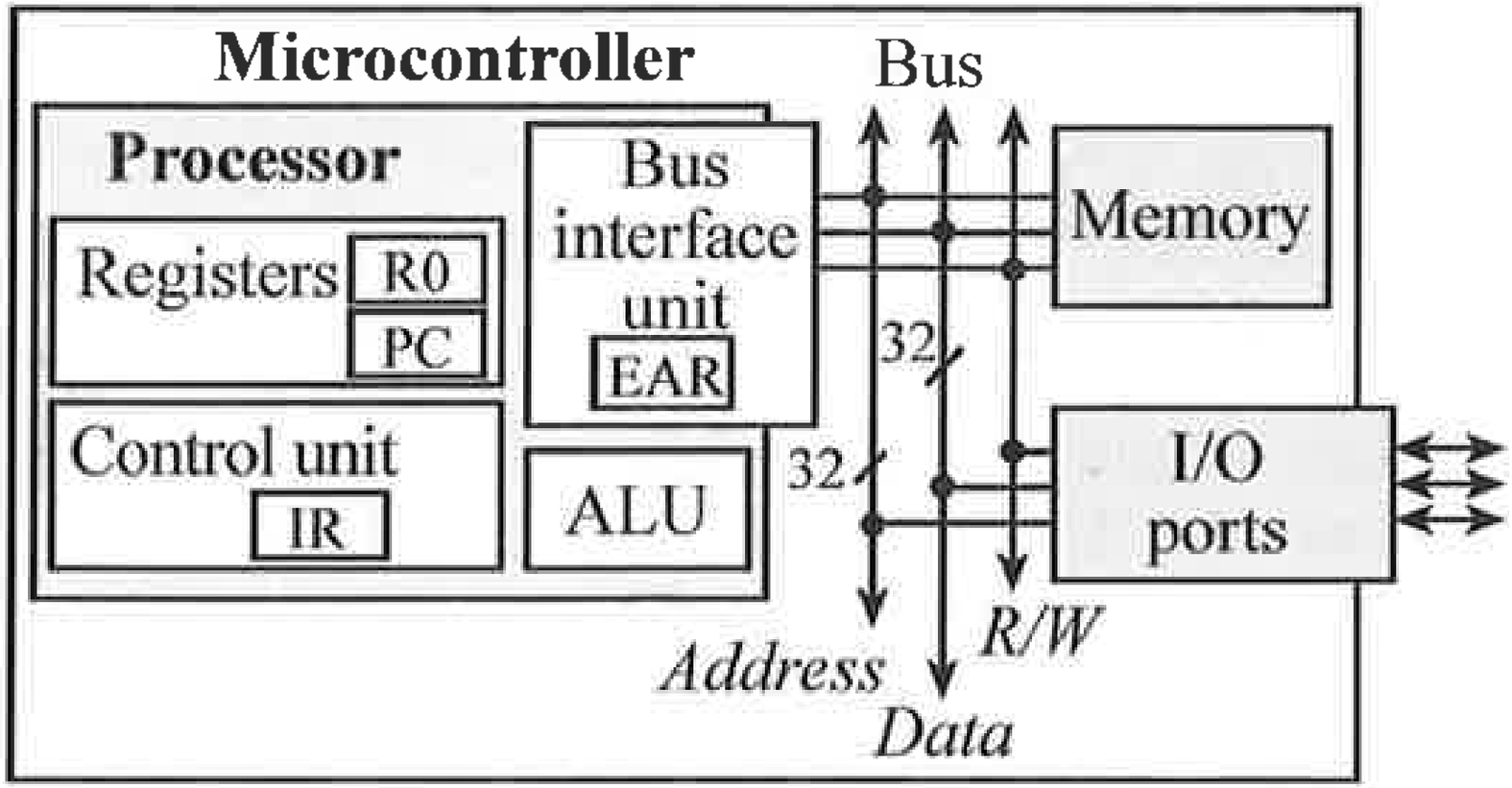
1. R flag set; processor sends out addr
2. ram gets request; sends data at address along bus to processor

W: 0x11223344 to addr. 0x2000.0004



1. W flag set; processor sends out addr and data along appropriate buses
2. ram gets data and addr; data stored at address

execution: R/W from memory



Phase	Function	R/W	Address	Comment
1	**Op code fetch**	**read**	**PC**	**Put op code into IR**
2	Decode instruction	none		Increment PC by 2 or by 4
3	Evaluation address	none		Determine EAR
4	**Data read**	**read**	**SP,EAR**	**Data passes through ALU,**
5	Free cycle			ALU operations
6	**Data store**	**write**	**SP,EAR**	**Results stored in memory**

Assembly I

ECE 3710

If toast always lands butter-side down, and cats always land on their feet, what happens if you strap toast on the back of a cat and drop it?

- Steven Wright

anatomy of an assembly instruction

why you're doing
what you're doing

action/operation

```
[label] mnemonic [operands] [;comment]
```

subroutine,
data, etc.

what action operates on
(immediates, registers, and memory)

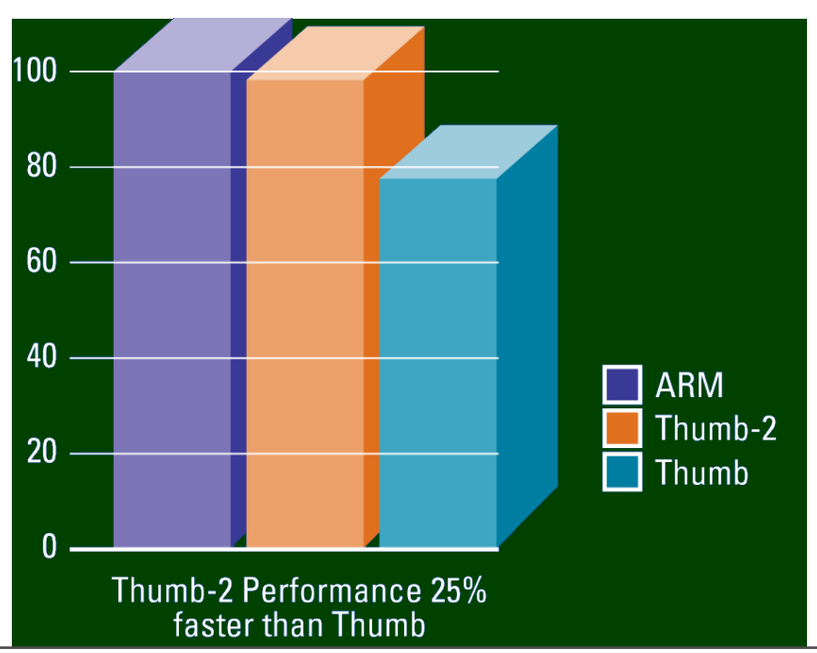
[ ] => optional

details of Cortex M3 assembly

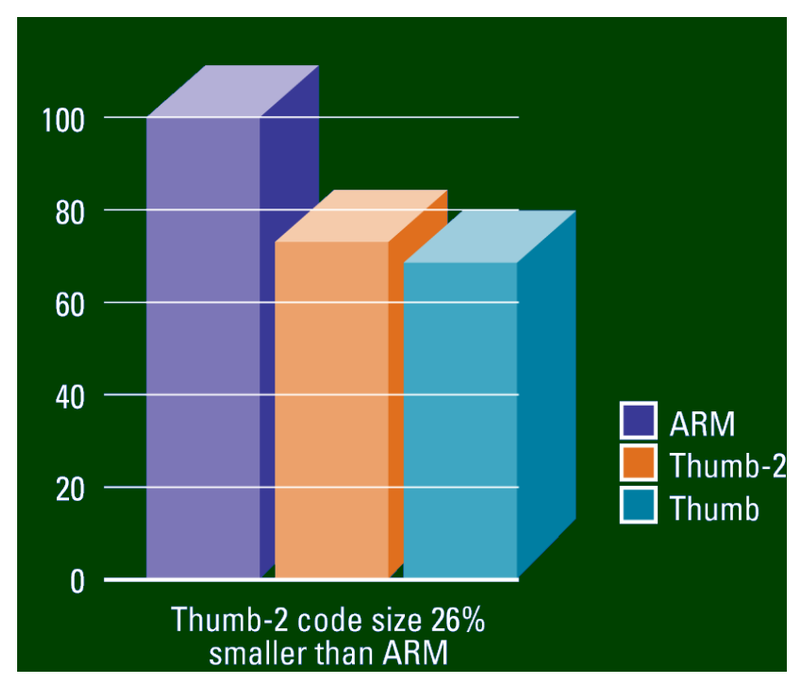
this is complicated:

1. each instruction has many options
2. a mix of 32- and 16-bit instructions

speed:



code density:



complexity inspires this:

???:

```
<op>{size} Rd, [Rn, +/-Rm {, <opsh>}]{!}
```

key to THUMB2 instructions:

`Ra Rd Rm Rn Rt:`	represent 32-bit registers
`value:`	any 32-bit value: signed, unsigned, or address
`{S}:`	if S is present, instruction will set condition codes
`#im12:`	any value from 0 to 4095
`#im16:`	any value from 0 to 65535
`{Rd,}:`	if Rd is present Rd is destination, otherwise Rn
`#n:`	any value from 0 to 31
`#off:`	any value from -255 to 4095
`label:`	any address within the ROM of the microcontroller
`op2:`	the value generated by <op2> (the flexible operator)

clear, eh?

a substantial number of you?

don’t leave me now... it will get better

combing ARM keywords

example:

```
ADD{COND}{S} {Rd, } Rn, #imm12 ;Rd = Rn + #imm12
```

mnemonic

talk more about these later

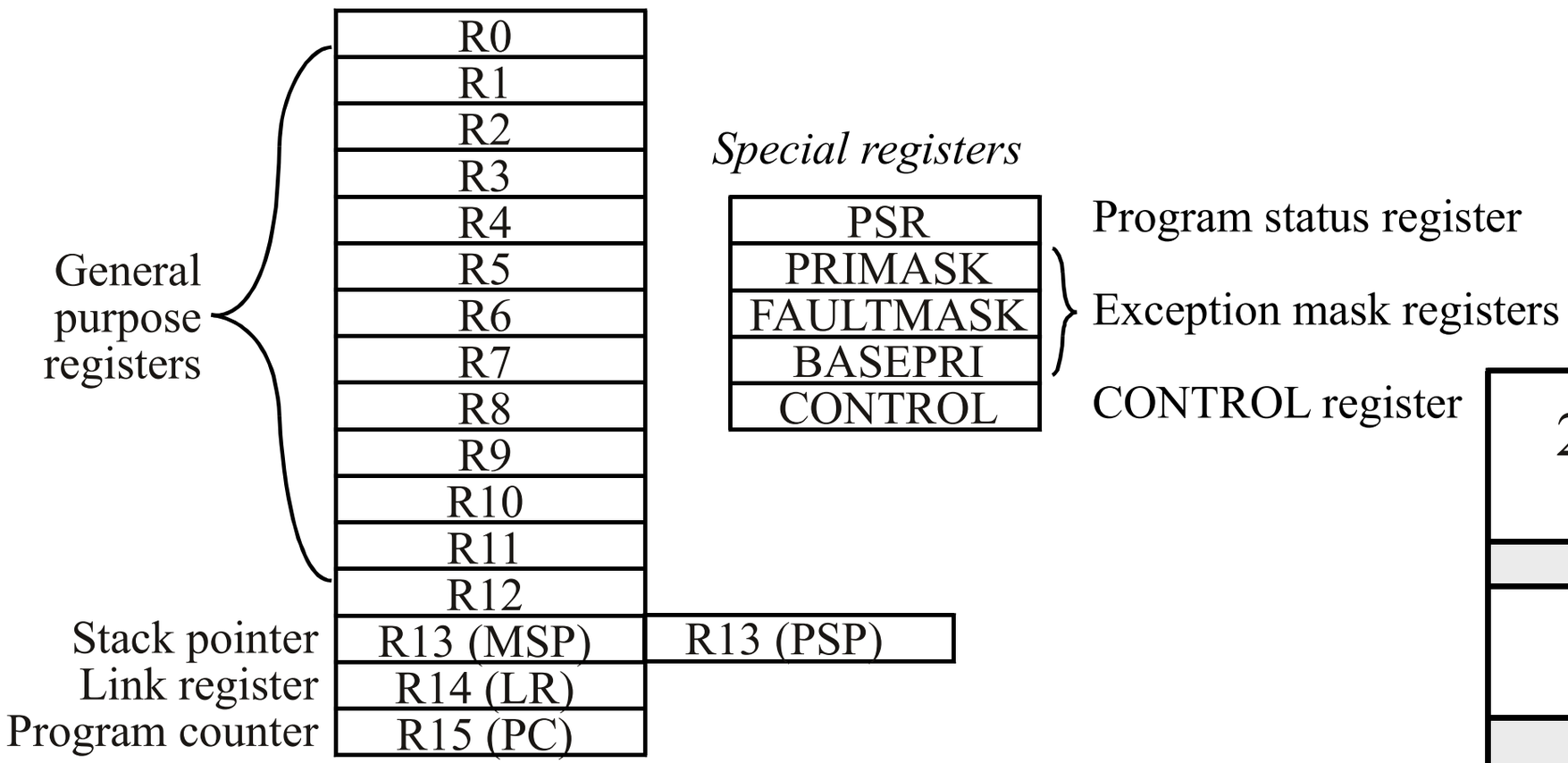
(focus on just mnemonic)

you will want a reference

(print PDF and keep with you)

what are registers and ram?

dedicated:

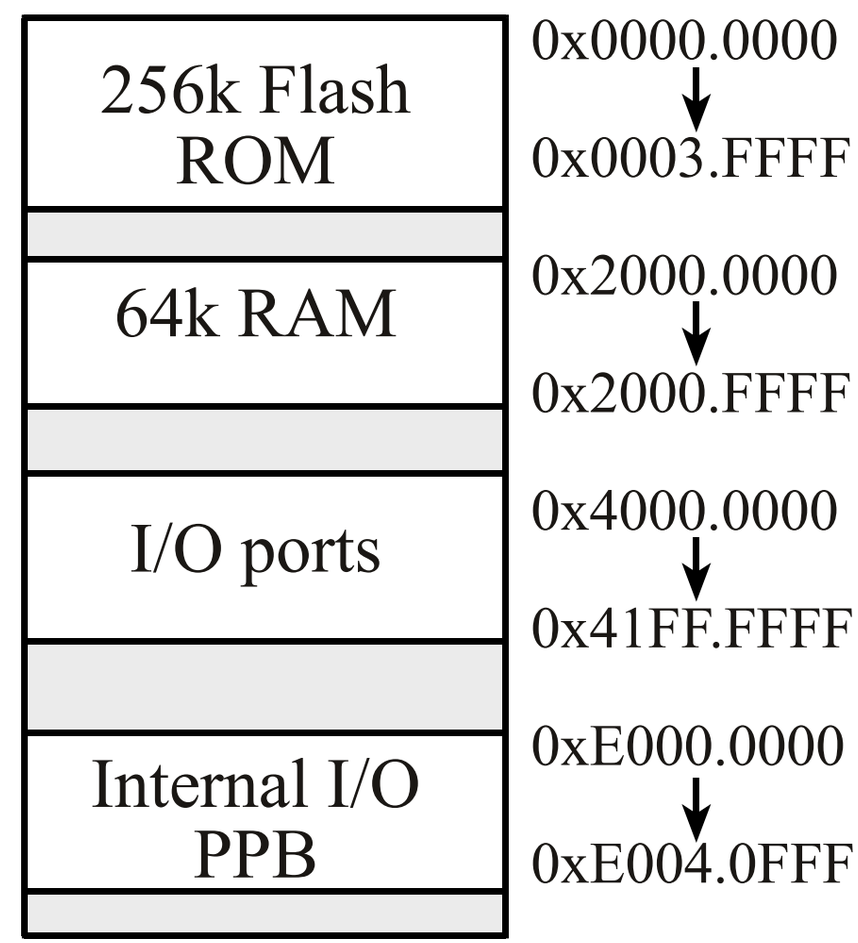


there are more...

(banked out)

memory:

(uC specific)



putting/moving something to register or memory

```
mov R0,#0xA ; R0 = 0xA
LDR R1,=0x20000000 ; R1 = 0x20000000
                   ;(this is a pseudo-instruction,
                   ;use for >#im16)
STR R0,[R1] ;R0 in what R1 points at:
            ;mem(0x20000000) = R0 = 0xA
STR R0,[R1,#0x18] ;R0 in what R1 + 0x18
                  ;points at:
                  ;mem(0x20000018)=0xA
LDR R3,[R1] ; R3 = mem(0x20000000)
MVN R0,#0x0 ;R0 = 0xFFFFFFFF
```

case *insensitive*, pay attention to *spaces*...also there's more

Move data	Move		`MOV{S} Rd, <Operand2>`	N Z C	Rd := Operand2 See also Shift instructions	N
	NOT		`MVN{S} Rd, <Operand2>`	N Z C	Rd := 0xFFFFFFFF EOR Operand2	N
	top	T2	`MOVT Rd, #<imm16>`		Rd[31:16] := imm16, Rd[15:0] unaffected, imm16 range 0-65535	
	wide	T2	`MOV Rd, #<imm16>`		Rd[15:0] := imm16, Rd[31:16] = 0, imm16 range 0-65535	
	40-bit accumulator to register	XS	`MRA RdLo, RdHi, Ac`		RdLo := Ac[31:0], RdHi := Ac[39:32]	
	register to 40-bit accumulator	XS	`MAR Ac, RdLo, RdHi`		Ac[31:0] := RdLo, Ac[39:32] := RdHi	

Single data item loads and stores		§	Assembler	Action if <op> is LDR	Action if <op> is STR	Notes
Load or store word, byte or halfword	Immediate offset		`<op>{size}{T} Rd, [Rn {, #<offset>}]{!}`	Rd := [address, size]	[address, size] := Rd	1, N
	Post-indexed, immediate		`<op>{size}{T} Rd, [Rn], #<offset>`	Rd := [address, size]	[address, size] := Rd	2
	Register offset		`<op>{size} Rd, [Rn, +/-Rm {, <opsh>}]{!}`	Rd := [address, size]	[address, size] := Rd	3, N
	Post-indexed, register		`<op>{size}{T} Rd, [Rn], +/-Rm {, <opsh>}`	Rd := [address, size]	[address, size] := Rd	4
	PC-relative		`<op>{size} Rd, <label>`	Rd := [label, size]	Not available	5, N
Load or store doubleword	Immediate offset	5E	`<op>D Rd1, Rd2, [Rn {, #<offset>}]{!}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	Post-indexed, immediate	5E	`<op>D Rd1, Rd2, [Rn], #<offset>`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	Register offset	5E	`<op>D Rd1, Rd2, [Rn, +/-Rm {, <opsh>}]{!}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	Post-indexed, register	5E	`<op>D Rd1, Rd2, [Rn], +/-Rm {, <opsh>}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	PC-relative	5E	`<op>D Rd1, Rd2, <label>`	Rd1 := [label], Rd2 := [label + 4]	Not available	8, 9

Other memory operations		§	Assembler	Action	Notes
Load multiple	Block data load		`LDM{IA|IB|DA|DB} Rn{!}, <reglist-PC>`	Load list of registers from [Rn]	N, I
	return (and exchange)		`LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>`	Load registers, PC := [address][31:1] (§ 5T: Change to Thumb if [address][0] is 1)	I
	and restore CPSR		`LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>^`	Load registers, branch (§ 5T: and exchange), CPSR := SPSR. Exception modes only.	I
	User mode registers		`LDM{IA|IB|DA|DB} Rn, <reglist-PC>^`	Load list of User mode registers from [Rn]. Privileged modes only.	I
Pop			`POP <reglist>`	Canonical form of `LDM SP!, <reglist>`	N
Load exclusive	Semaphore operation	6	`LDREX Rd, [Rn]`	Rd := [Rn], tag address as exclusive access. Outstanding tag set if not shared address. Rd, Rn not PC.	
	Halfword or Byte	6K	`LDREX{H|B} Rd, [Rn]`	Rd[15:0] := [Rn] or Rd[7:0] := [Rn], tag address as exclusive access. Outstanding tag set if not shared address. Rd, Rn not PC.	
	Doubleword	6K	`LDREXD Rd1, Rd2, [Rn]`	Rd1 := [Rn], Rd2 := [Rn+4], tag addresses as exclusive access Outstanding tags set if not shared addresses. Rd1, Rd2, Rn not PC.	9
Store multiple	Push, or Block data store		`STM{IA|IB|DA|DB} Rn{!}, <reglist>`	Store list of registers to [Rn]	N, I
	User mode registers		`STM{IA|IB|DA|DB} Rn{!}, <reglist>^`	Store list of User mode registers to [Rn]. Privileged modes only.	I
Push			`PUSH <reglist>`	Canonical form of `STMDB SP!, <reglist>`	N
Store exclusive	Semaphore operation	6	`STREX Rd, Rm, [Rn]`	If allowed, [Rn] := Rm, clear exclusive tag, Rd := 0. Else Rd := 1. Rd, Rm, Rn not PC.	
	Halfword or Byte	6K	`STREX{H|B} Rd, Rm, [Rn]`	If allowed, [Rn] := Rm[15:0] or [Rn] := Rm[7:0], clear exclusive tag, Rd := 0. Else Rd := 1 Rd, Rm, Rn not PC.	
	Doubleword	6K	`STREXD Rd, Rm1, Rm2, [Rn]`	If allowed, [Rn] := Rm1, [Rn+4] := Rm2, clear exclusive tags, Rd := 0. Else Rd := 1 Rd, Rm1, Rm2, Rn not PC.	9
Clear exclusive		6K	`CLREX`	Clear local processor exclusive tag	C

know by Monday

adding something to a register

```
;ADD Rn,#imm12
ADD R0,#0xA
;ADD Rd,Rn,#imm12
ADD R3,R0,#0x2 ;R3 = R0+0x2
; ADD Rd, Rn, <op2>
...
```

<op2>: the flexible second operator

ASR,LSL,LSR,ROR,RRX

can take on these forms:

1. register (with or without a shift)
2. constant produced by (not) shifting `#imm8` (assembler does this for you)
3. constant: `0x00XY00XY`
4. constant: `0xXY00XY00`
5. constant: `0xXYXYXYXY`

addition and <op2>

register, without shift

```
; ADD Rd Rn <op2>
ADD R2,R0,R1 ;R2 = R0 + R1
ADD R4,R0,#0x20000000 ;R4=R0+#0x20000000
ADD R5,R0,R1,LSL #1 ;R5 = R0 + 2*R1
```

register, with shift

#imm8 w/shift

store result in R0

20

Q: x^2*(y+z) = ?

10

30

write assembly
to calculate this