Division

Lecture Objectives:

1) Perform binary division of two numbers.
2) Define dividend, divisor, quotient, and remainder.
3) Explain how division is accomplished in computer hardware.
4) Construct a simple program which uses MIPS integer multiplication and division

Friday $ fun :-)

⇒ Exam

One 8.5 x 11 sheet of notes.

- Review sheet online.

⇒ 5 minutes going over sheet & ask questions

Questions:

Harvard VS Von Neuman

DSP's

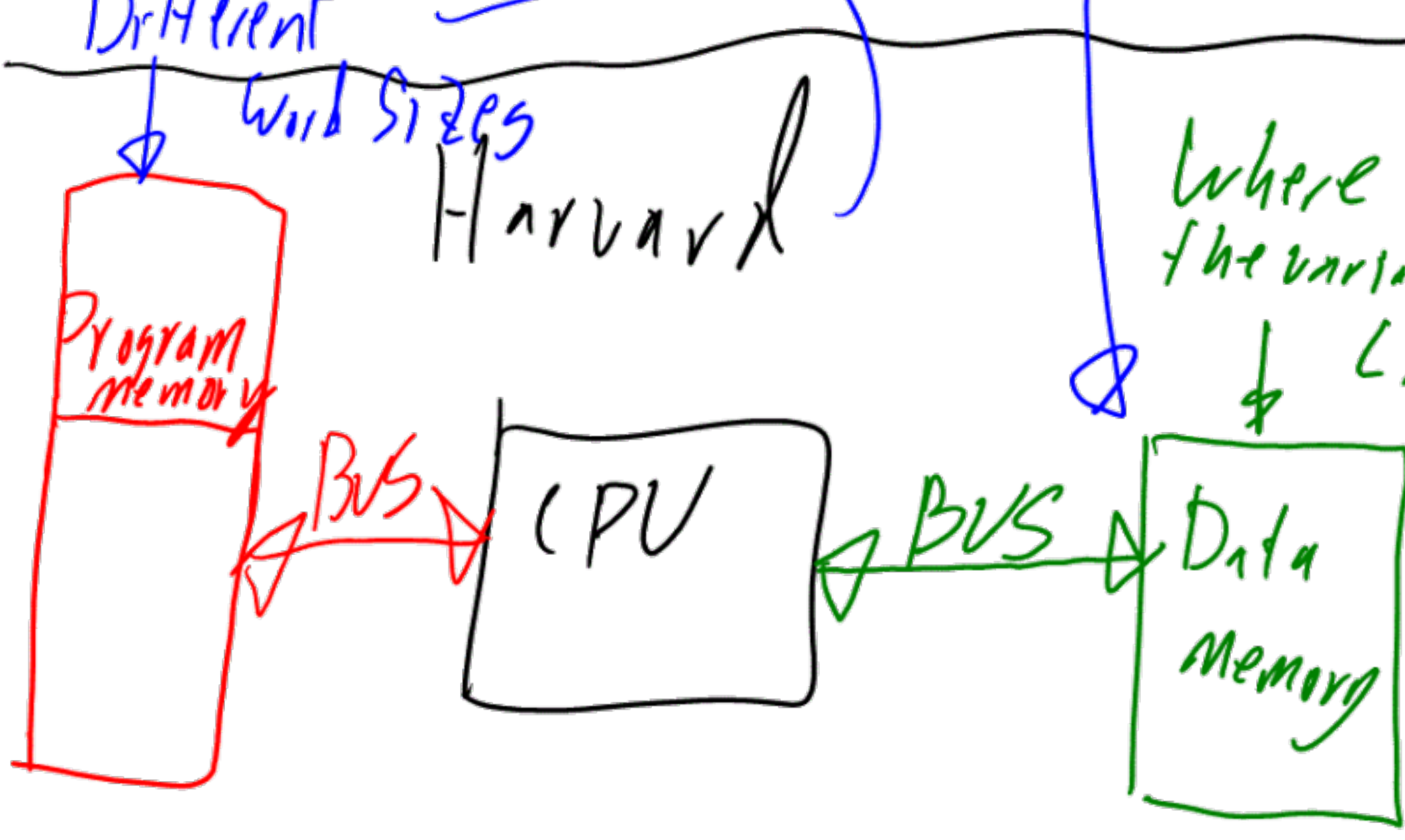
ATMEGA32

Different

Word Sizes

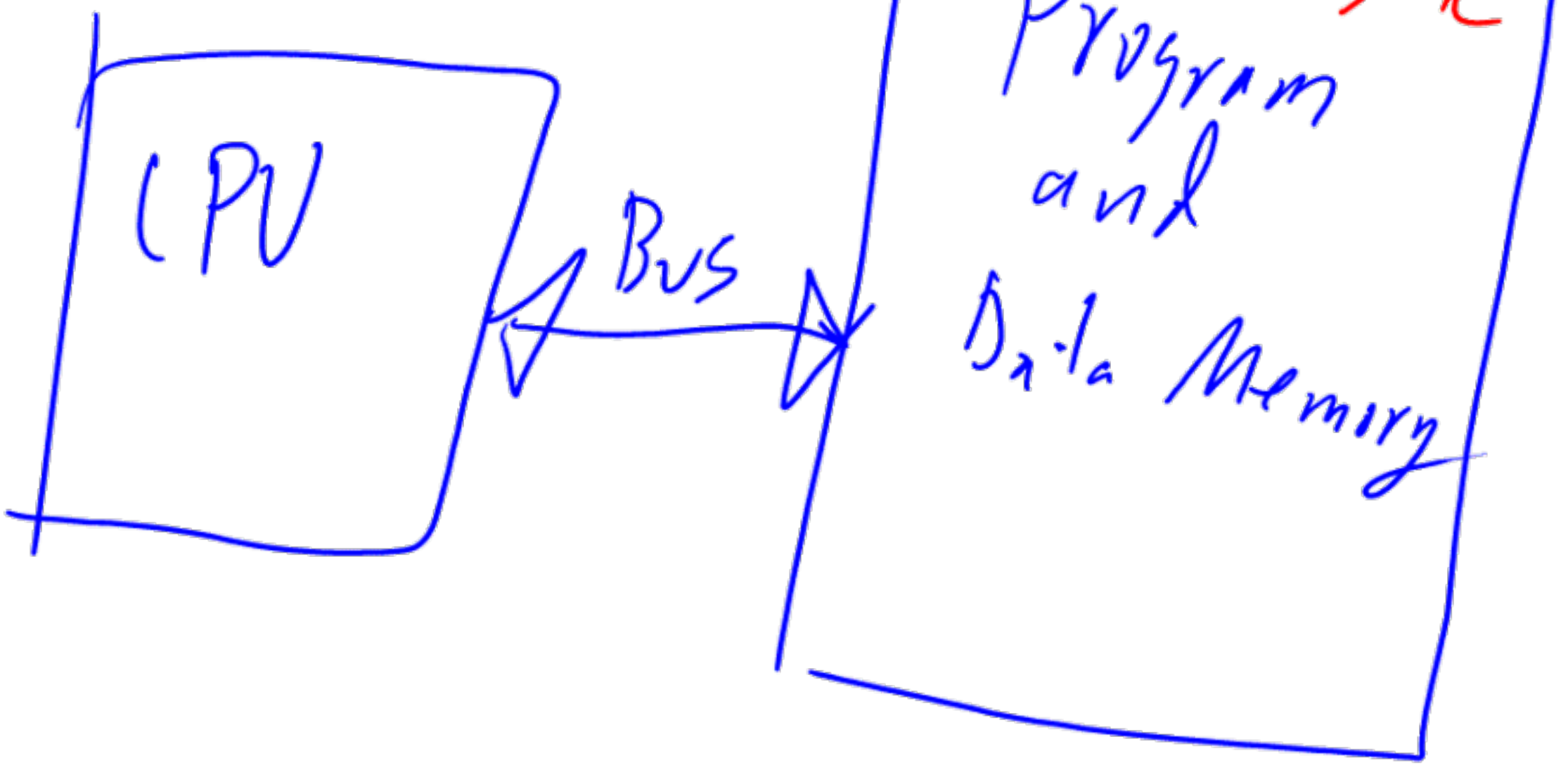
Harvard

Where the variables Live



Von Neumann

MIPS, ARM, Intel, Motorola, almost everyone



-5 ⇒ 2's Complement

|5| ⇒ 0000 0101 ⇒ 5

⇒ Complement it ⇒ 1111 1010

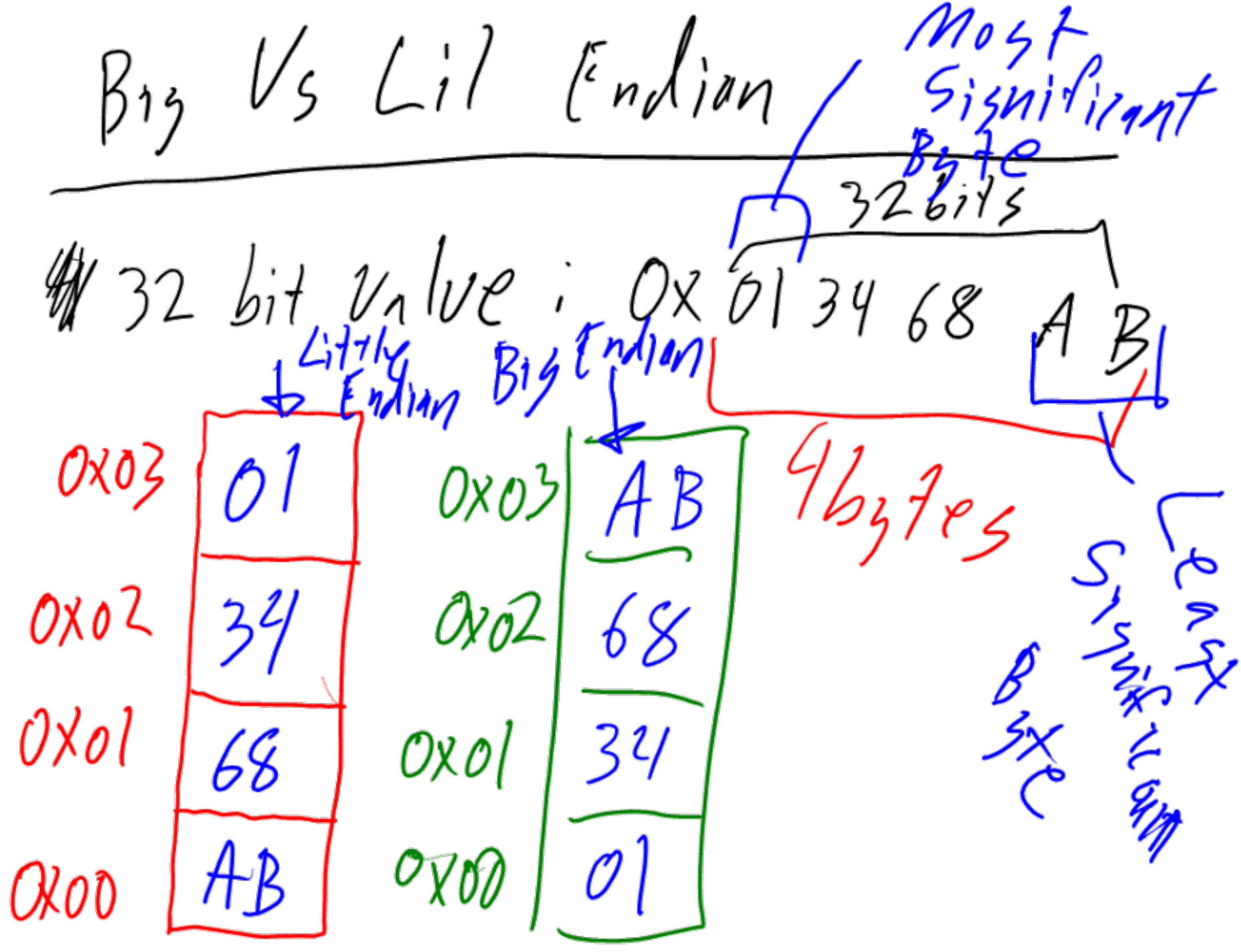
⇒ Add one 1111 1010 + 1 ⇒ 1111 1011 ⇒ -5 in 2's Complement

Equations $\Rightarrow$ Good to write on note sheet.

Packet/sheet of MCPs
instructions will be provided.

Big Vs Lil Endian

// 32 bit value : 0x01 34 68 AB



NOR:

Always @ zero

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

A

S

Same thing

NOT

+ Pseudo instruction

A	C
0	1
1	0

Definitions (Pg 237)

- Dividend
  - The first operand of a divide operation
- Divisor
  - The second operand of a divide operation
- Quotient
  - The final result of a divide operation
- Remainder
  - The second result to a division operation

→ % Modulus

Division (Long Approach)

- Solve the following

$$8\overline{)740}$$

092 R 4

Quotient

Remainder

72

20

16

4

Division Algorithm

- Check for 0 divisor
- Long division approach
  - If divisor ≤ dividend bits
    - 1 bit in quotient, subtract
  - Otherwise
    - 0 bit in quotient, bring down next dividend bit
- Restoring division
  - Do the subtract, and if remainder goes < 0, add divisor back
- Signed division
  - Divide using absolute values
  - Adjust sign of quotient and remainder as required

$$2\overline{)18}\ \ 9$$

Unsigned #'s

$$10\overline{)10010}\ \ 18$$

$$\begin{array}{r} 10 \\ \hline 00010 \\ 10 \\ \hline 00 \end{array}$$

01001 + 9

Quotient =>
Remainder =>
} Both placed in Registers

5 | 27 → 16 8 2 1
4 | 1 1 0 1 1
1 | 0 1 0 1

5 R 2

101 R 010

0 0 1 0 1
0101 | 1011
101
0011
101
010

Time is related to how big the H's are.

Faster Division

- Can't use parallel hardware as in multiplier
  - Subtraction is conditional on sign of remainder
- Faster dividers (e.g. SRT devision) generate multiple quotient bits per step
  - Still require multiple steps

MIPS Division

- Use HI/LO registers for result
  - HI: 32-bit remainder
  - LO: 32-bit quotient
- Instructions
  - `div rs, rt` / `divu rs, rt`
  - No overflow or divide-by-0 checking
    - Software must perform checks if required
  - Use `mfhi`, `mflo` to access result

Instructions we've talked about thusfar

MIPS assembly language

Category	Instruction	Example		Meaning	Comments
Arithmetic	add	`add`	`$s1,$s2,$s3`	`$s1 = $s2 + $s3`	Three operands; overflow detected
	subtract	`sub`	`$s1,$s2,$s3`	`$s1 = $s2 – $s3`	Three operands; overflow detected
	add immediate	`addi`	`$s1,$s2,100`	`$s1 = $s2 + 100`	+ constant; overflow detected
	add unsigned	`addu`	`$s1,$s2,$s3`	`$s1 = $s2 + $s3`	Three operands; overflow undetected
	subtract unsigned	`subu`	`$s1,$s2,$s3`	`$s1 = $s2 – $s3`	Three operands; overflow undetected
	add immediate unsigned	`addiu`	`$s1,$s2,100`	`$s1 = $s2 + 100`	+ constant; overflow undetected
	move from coprocessor register	`mfc0`	`$s1,$epc`	`$s1 = $epc`	Copy Exception PC + special regs
	multiply	`mult`	`$s2,$s3`	Hi, Lo = `$s2 × $s3`	64-bit signed product in Hi, Lo
	multiply unsigned	`multu`	`$s2,$s3`	Hi, Lo = `$s2 × $s3`	64-bit unsigned product in Hi, Lo
	divide	`div`	`$s2,$s3`	Lo = `$s2 / $s3`, Hi = `$s2 mod $s3`	Lo = quotient, Hi = remainder
	divide unsigned	`divu`	`$s2,$s3`	Lo = `$s2 / $s3`, Hi = `$s2 mod $s3`	Unsigned quotient and remainder
	move from Hi	`mfhi`	`$s1`	`$s1 = Hi`	Used to get copy of Hi
	move from Lo	`mflo`	`$s1`	`$s1 = Lo`	Used to get copy of Lo
Data transfer	load word	`lw`	`$s1,20($s2)`	`$s1 = Memory[$s2 + 20]`	Word from memory to register
	store word	`sw`	`$s1,20($s2)`	`Memory[$s2 + 20] = $s1`	Word from register to memory
	load half unsigned	`lhu`	`$s1,20($s2)`	`$s1 = Memory[$s2 + 20]`	Halfword memory to register
	store half	`sh`	`$s1,20($s2)`	`Memory[$s2 + 20] = $s1`	Halfword register to memory
	load byte unsigned	`lbu`	`$s1,20($s2)`	`$s1 = Memory[$s2 + 20]`	Byte from memory to register
	store byte	`sb`	`$s1,20($s2)`	`Memory[$s2 + 20] = $s1`	Byte from register to memory
	load linked word	`ll`	`$s1,20($s2)`	`$s1 = Memory[$s2 + 20]`	Load word as 1st half of atomic swap
	store conditional word	`sc`	`$s1,20($s2)`	`Memory[$s2+20]=$s1;$s1=0 or 1`	Store word as 2nd half atomic swap
	load upper immediate	`lui`	`$s1,100`	`$s1 = 100 * 2^16`	Loads constant in upper 16 bits
Logical	AND	`AND`	`$s1,$s2,$s3`	`$s1 = $s2 & $s3`	Three reg. operands; bit-by-bit AND
	OR	`OR`	`$s1,$s2,$s3`	`$s1 = $s2 | $s3`	Three reg. operands; bit-by-bit OR
	NOR	`NOR`	`$s1,$s2,$s3`	`$s1 = ~ ($s2 |$s3)`	Three reg. operands; bit-by-bit NOR
	AND immediate	`ANDi`	`$s1,$s2,100`	`$s1 = $s2 & 100`	Bit-by-bit AND with constant
	OR immediate	`ORi`	`$s1,$s2,100`	`$s1 = $s2 | 100`	Bit-by-bit OR with constant
	shift left logical	`sll`	`$s1,$s2,10`	`$s1 = $s2 << 10`	Shift left by constant
	shift right logical	`srl`	`$s1,$s2,10`	`$s1 = $s2 >> 10`	Shift right by constant
Conditional branch	branch on equal	`beq`	`$s1,$s2,25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne`	`$s1,$s2,25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt`	`$s1,$s2,$s3`	if (`$s2 < $s3`) `$s1 = 1`; else `$s1 = 0`	Compare less than; two's complement
	set less than immediate	`slti`	`$s1,$s2,100`	if (`$s2 < 100`) `$s1 = 1`; else `$s1=0`	Compare < constant; two's complement
	set less than unsigned	`sltu`	`$s1,$s2,$s3`	if (`$s2 < $s3`) `$s1 = 1`; else `$s1=0`	Compare less than; natural numbers
	set less than immediate unsigned	`sltiu`	`$s1,$s2,100`	if (`$s2 < 100`) `$s1 = 1`; else `$s1 = 0`	Compare < constant; natural numbers
Unconditional jump	jump	`j`	2500	go to 10000	Jump to target address
	jump register	`jr`	`$ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal`	2500	`$ra` = PC + 4; go to 10000	For procedure call

Multiply Division Specific

Old

Old

Pointers

MSOE

In Class Example

- Lets write an assembly program which will perform the following:
  - Read two integer numbers from the user
  - Print out their product
  - Print out their quotient
  - Print out the remainder (If the numbers are not evenly divisible)