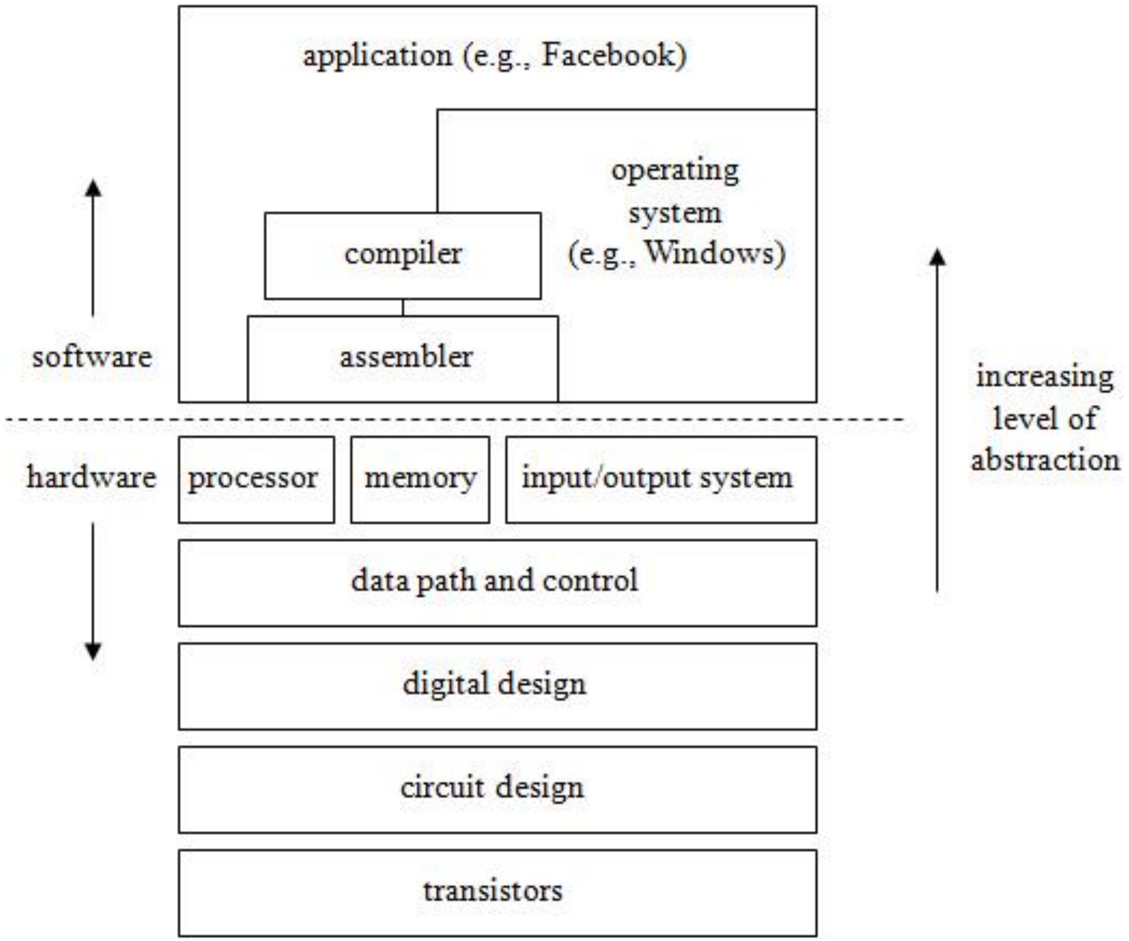
RISC/CISC- ARCHITECTURES

Computer Architecture

CISC/RISC

Computer levels



application (e.g., Facebook)

operating system (e.g., Windows)

compiler

assembler

software

hardware

processor

memory

input/output system

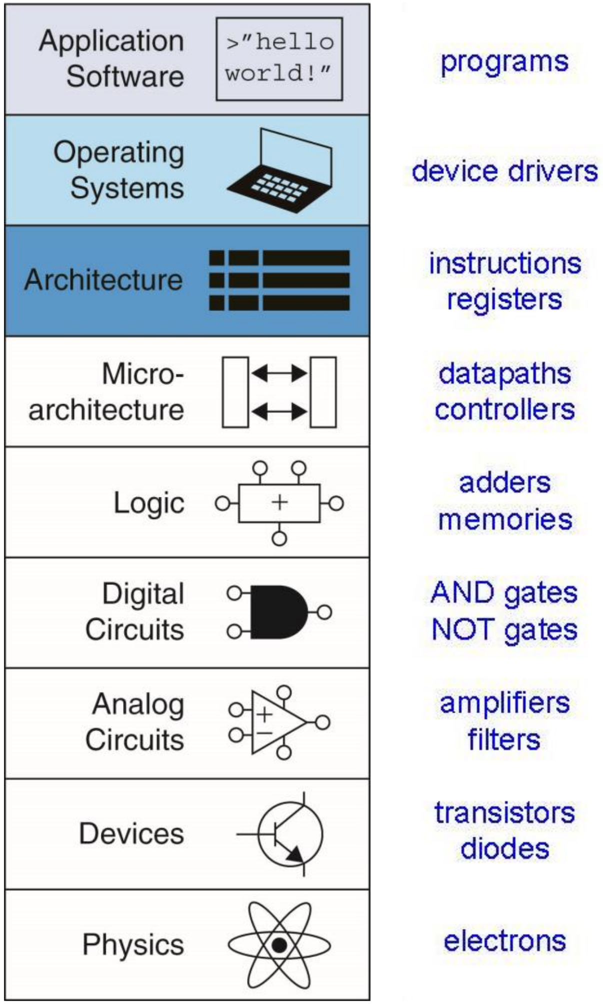
data path and control

digital design

circuit design

transistors

increasing level of abstraction



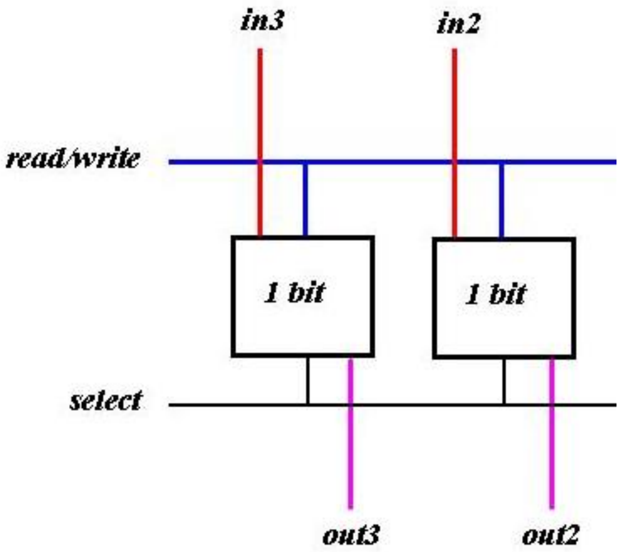
Application Software >"hello world!"	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

http://kpharri.files.wordpress.com/

My Add(2) Computer System

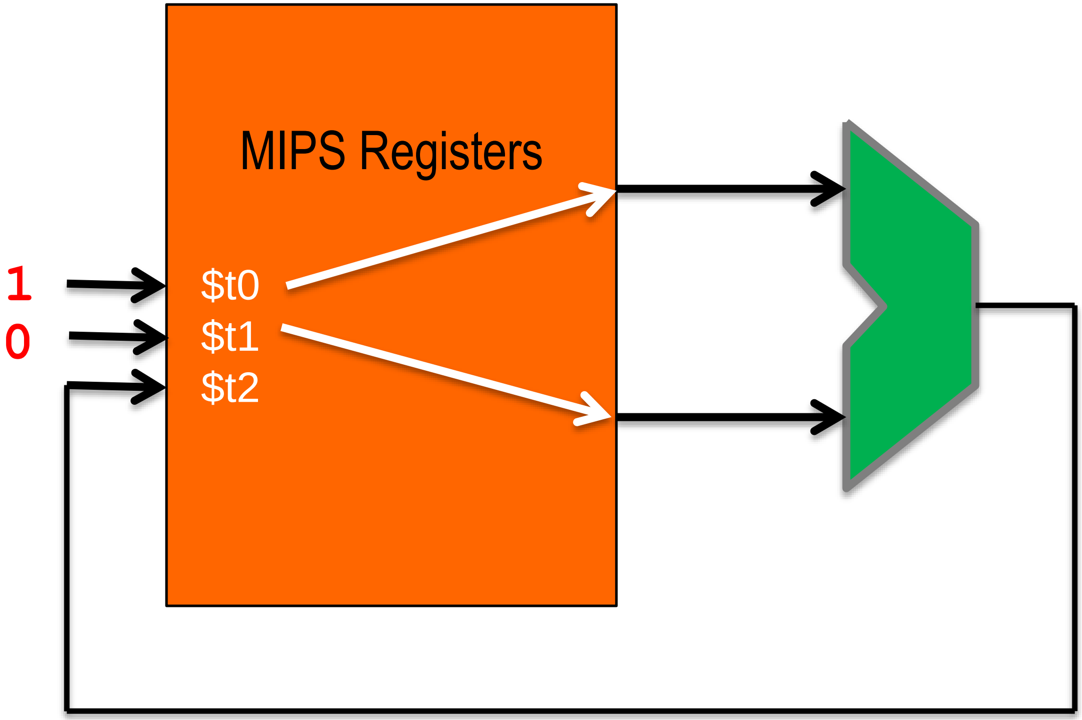
My Add(2) Computer System

- Read (Load from the memory) two numbers: **1** and **0**
- Store temporarily (to registers) the two numbers to be added



- Add the two numbers (Arithmetic Logic Unit)
- Store back temporarily (to a register) the result

My Add(2) Computer System



rs (First **s**ource **r**egister operand)
rt (Second source **r**egister operand)
rd (**D**estination **r**egister operand)

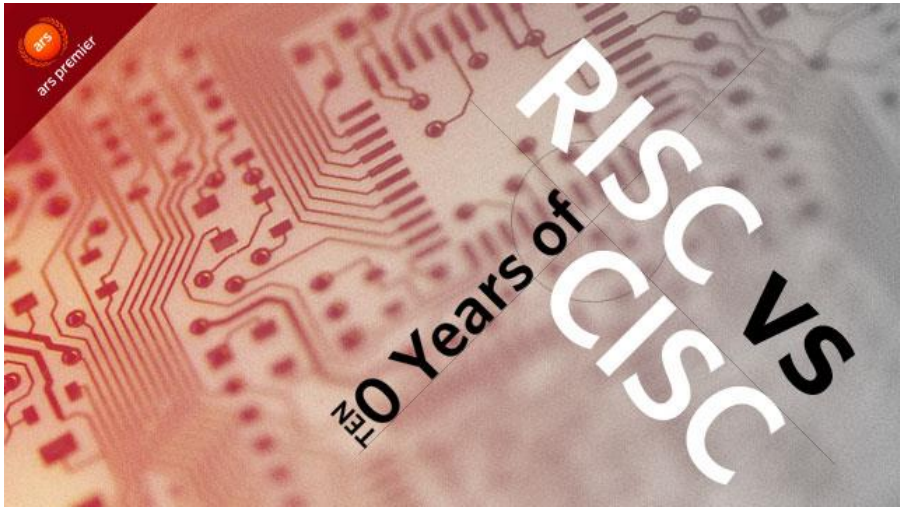
```
add rd, rs, rt
            1    0
```

We have …

- ALU
  - Add: 2 numbers …
- Memory
  - Load from Memory to Register-File
- Registers
  - Store temporarily
  - How many registers do we have in a real computer system?

Computer systems

- CISC (Complex Instruction set Computer)
  - – 16 registers (INTEL x86, …)
- RISC (Reduced Instruction set Computer)
  - – 32 general registers (MIPS, ARM, Arduino Duo, …)



CISC and RISC computers

	CISC	
	IBM370	80486
Instructions	**208**	**235**
Instruction size (bytes)	**2-6**	**1-12**
General registers	**16**	**8**
Developed	**1973**	**1989**

First CISC – IBM/360 (1964)

CISC and RISC computers

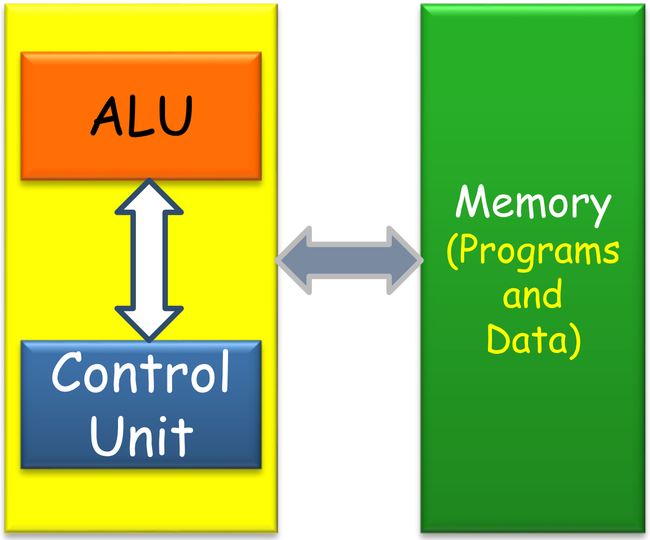
	CISC		RISC	
	IBM370	80486	SPARC	MIPS
Instructions	208	235	69	94
Instruction size (bytes)	2-6	1-12	4	4
General registers	16	8	40-580	32
Developed	1973	1989	1987	1991

First CISC – IBM/360 (1964)

First RISC – Burroughs B5000 (1963)

Von Newmann/Turing

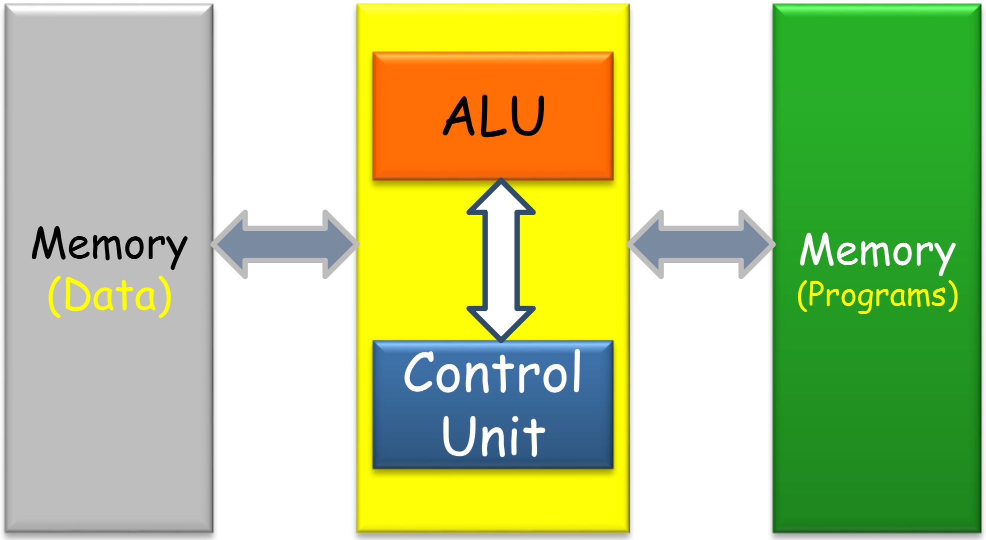
CISC



The von Neumann Architecture is named after the scientist John von Neumann (1942-1951)

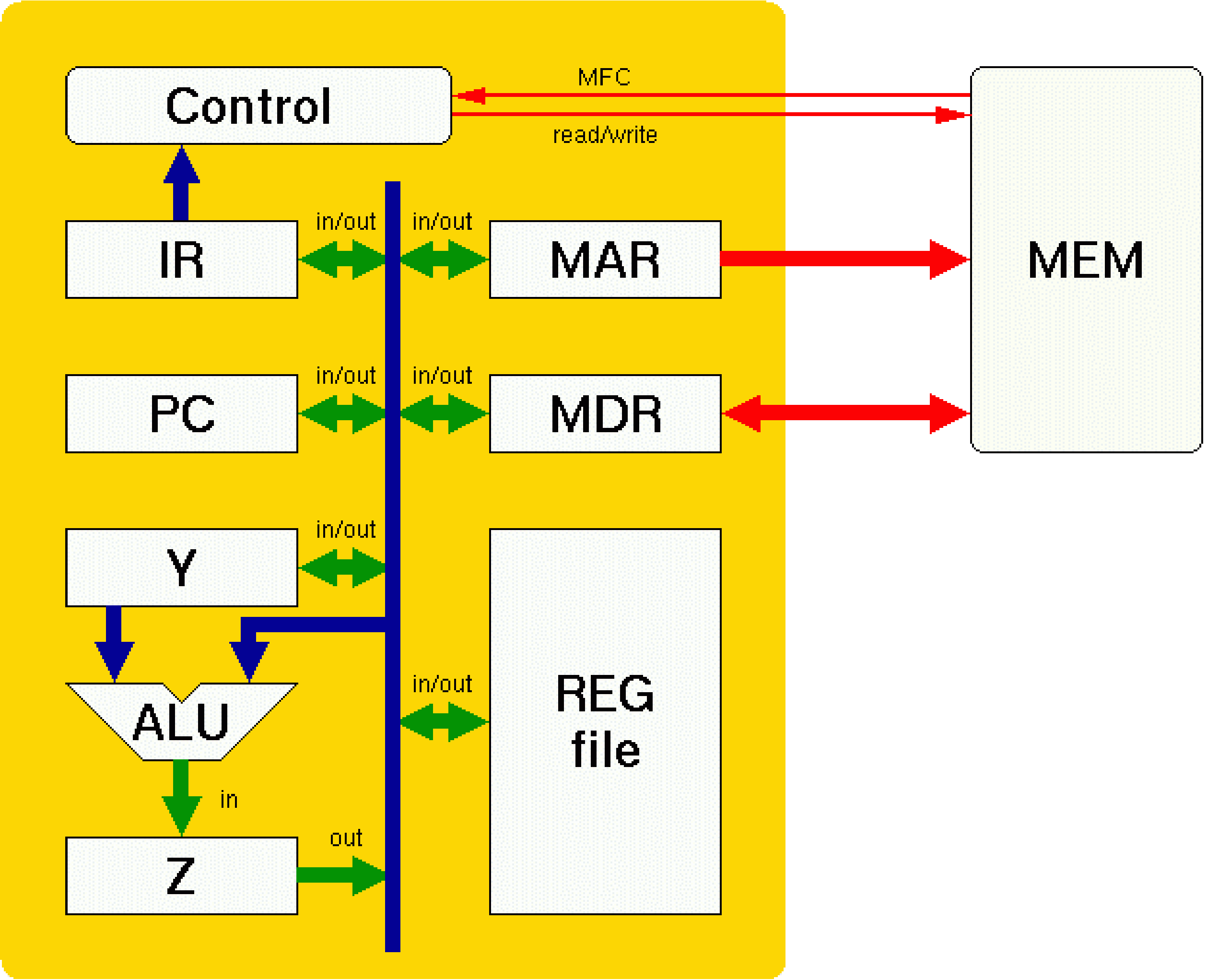
Harvard Architecture

RISC



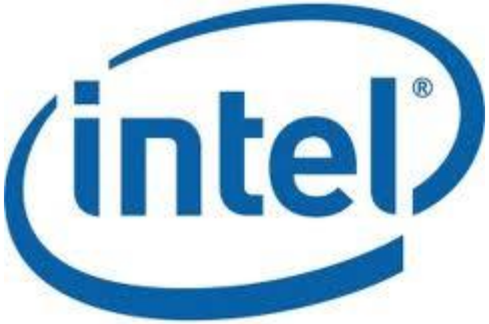
The name Harvard Architecture comes from the *Harvard Mark I* computer (1944)

CISC

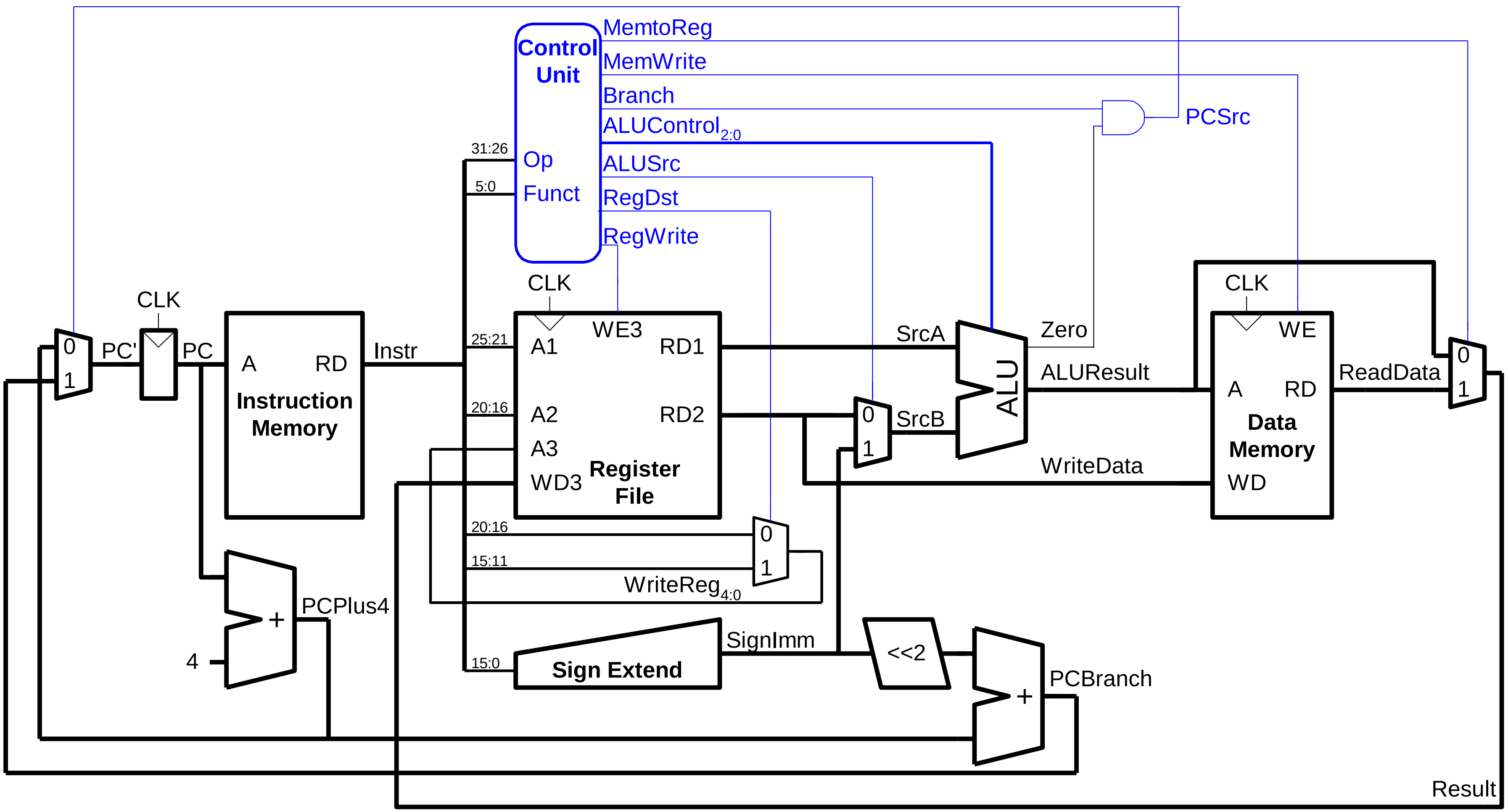


http://www.cise.ufl.edu/~mssz/CompOrg/Top-Level.html

CISC (CPUs)

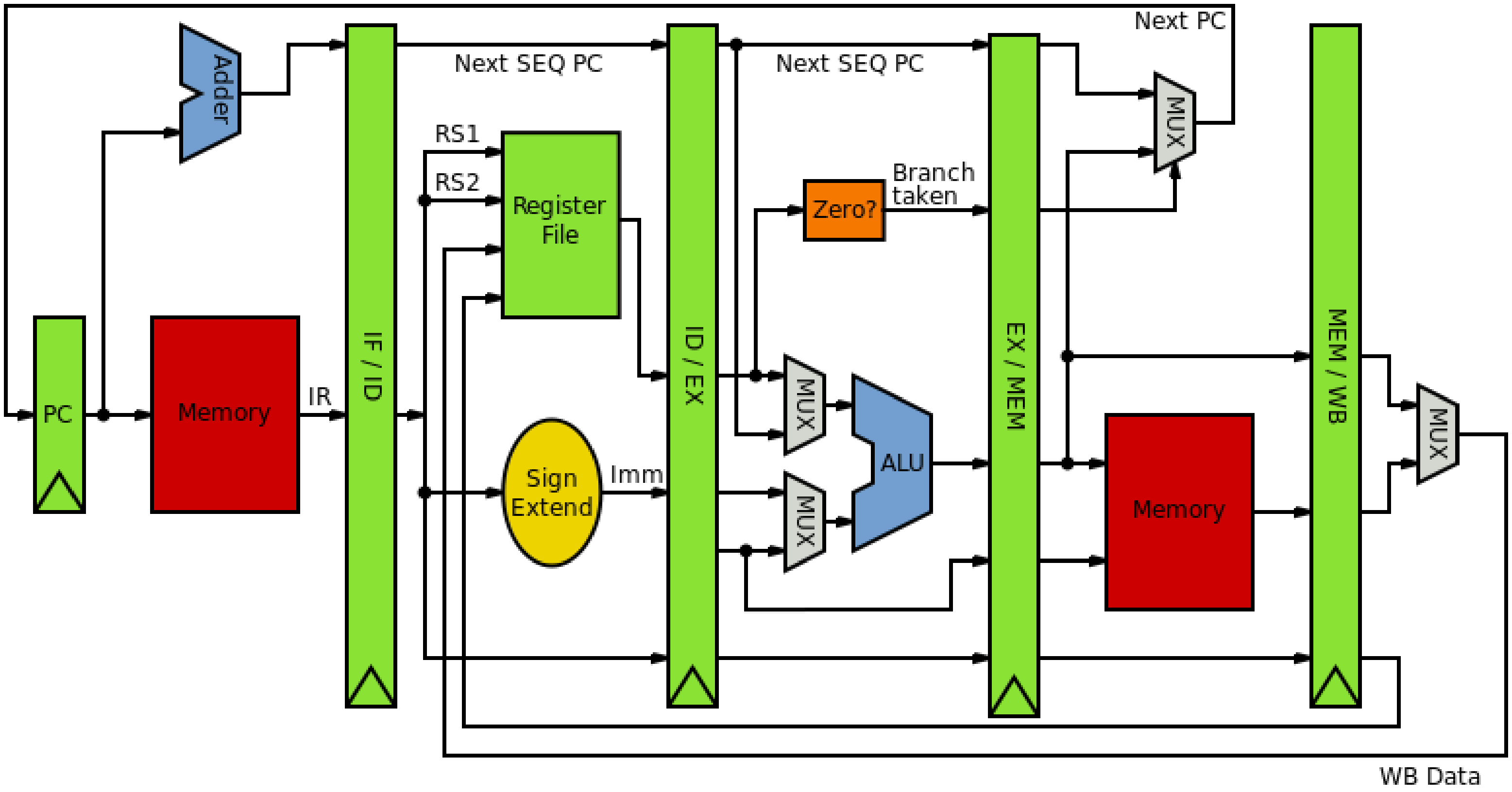


MIPS Architecture (Single Cycled)



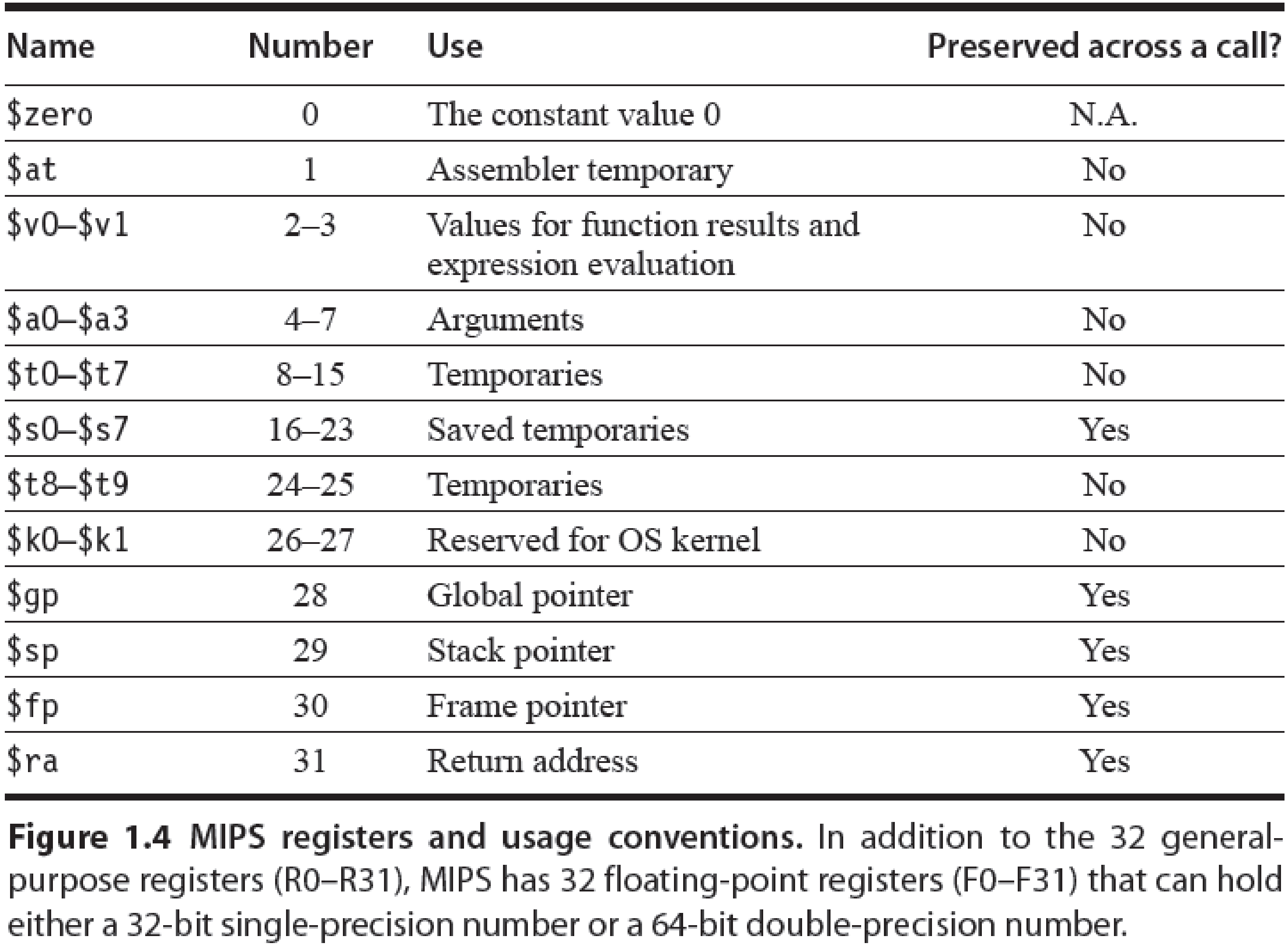
The MIPS – RISC Microprocessor

Instruction Fetch	Instruction Decode Register Fetch	Execute Address Calc.	Memory Access	Write Back
IF	ID	EX	MEM	WB



MIPS (**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages) is a **R**educed **I**nstruction **S**et **C**omputer (**RISC**)

RISC - Registers

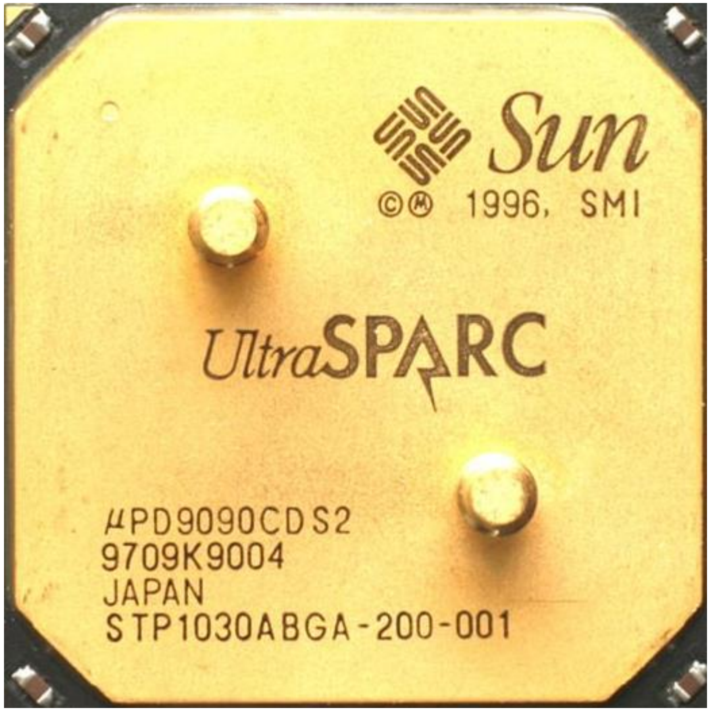


Name	Number	Use	Preserved across a call?
$zero	0	The constant value 0	N.A.
$at	1	Assembler temporary	No
$v0–$v1	2–3	Values for function results and expression evaluation	No
$a0–$a3	4–7	Arguments	No
$t0–$t7	8–15	Temporaries	No
$s0–$s7	16–23	Saved temporaries	Yes
$t8–$t9	24–25	Temporaries	No
$k0–$k1	26–27	Reserved for OS kernel	No
$gp	28	Global pointer	Yes
$sp	29	Stack pointer	Yes
$fp	30	Frame pointer	Yes
$ra	31	Return address	Yes

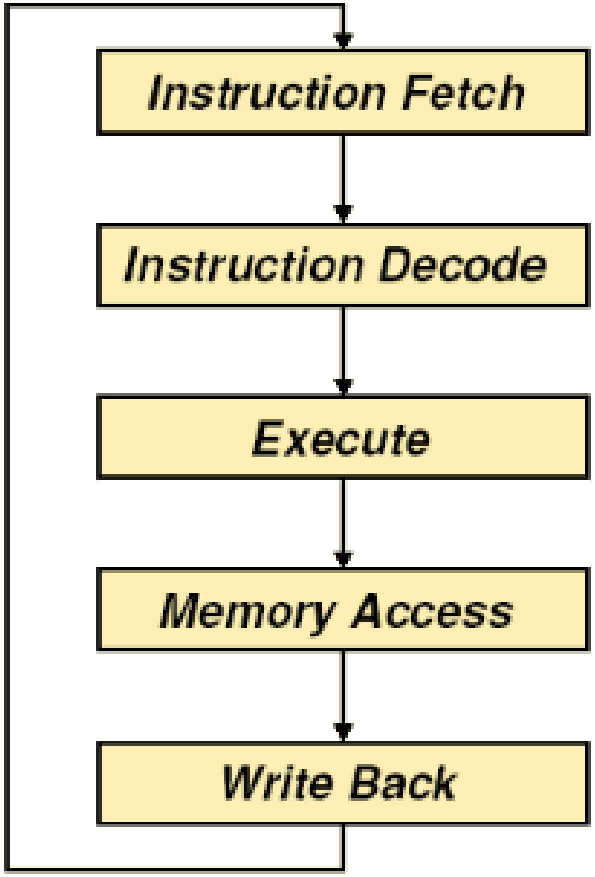
Figure 1.4 MIPS registers and usage conventions. In addition to the 32 general-purpose registers (R0–R31), MIPS has 32 floating-point registers (F0–F31) that can hold either a 32-bit single-precision number or a 64-bit double-precision number.

RISC (CPUs)

iPhone 3G S



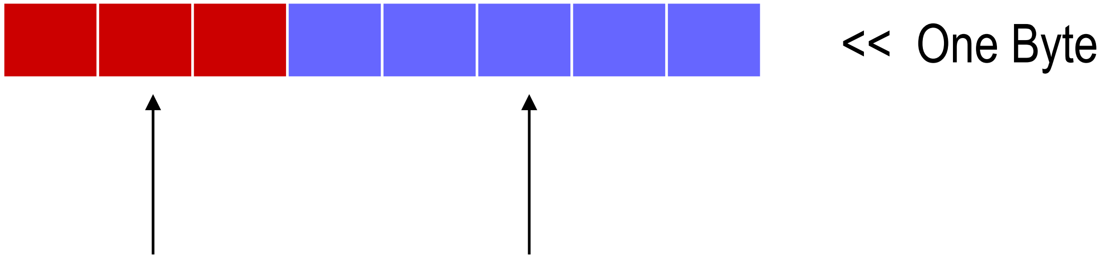
Each Instruction execution …



Next Instruction

- The CPU reads the next instruction from memory
  - It is placed in the **I**nstruction **R**egister (**IR**)
  - [The **P**rogram **C**ounter (**PC**) register holds the address of the current Instruction]
- The control unit then translates the instruction into the sequence of micro-operations necessary to implement the instruction.

Instruction …only “one-byte” long



<< One Byte

operations code (op-code)

Operand (address)

3-bits limit the number of instructions to $2^3 = 8$

5-bit limit the number of words to $2^5 = 32$

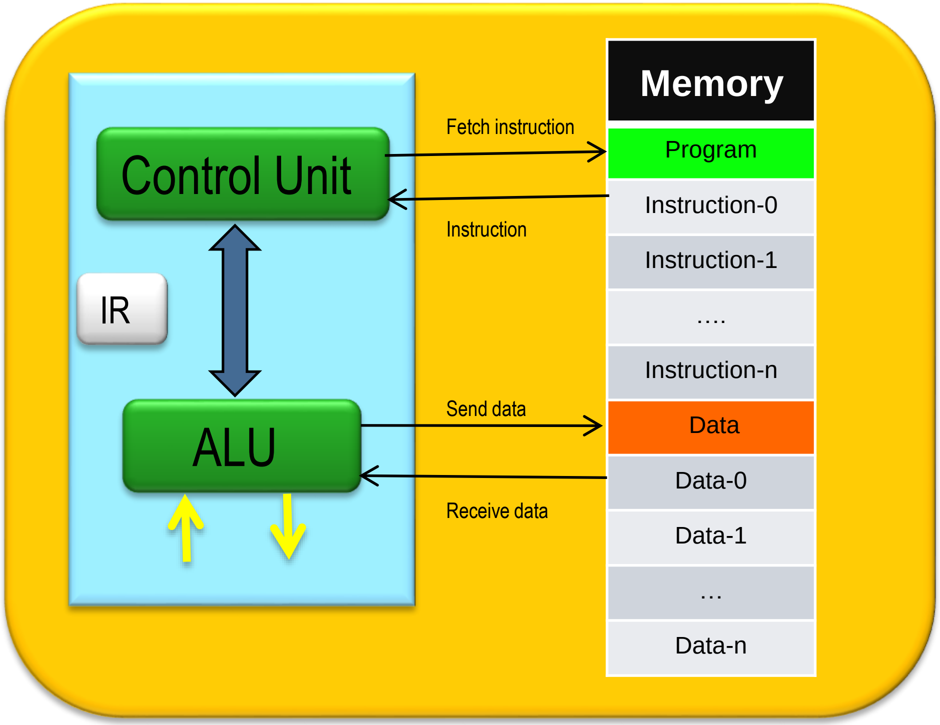
Usually the leftmost bit denotes the addressing mode

Each machine language instruction is composed of two parts:

1. Op-code
2. Operand (address)

Instructions = Program

The program consists of a sequence of instructions:
instruction-**0**, instruction-**1**, instruction-**2**, … <<< **order of execution**
Instruction Register: IR (Holds the memory address of the new (next) instruction)



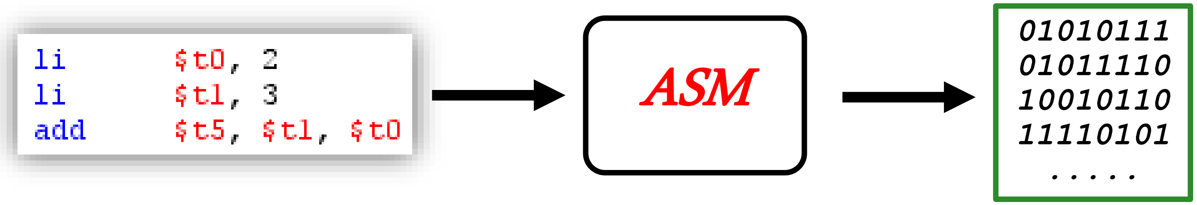
Instructions … Memory … Address

- The program consists of a sequence of instructions
- **Program Counter (PC) Register >>** holds the *address* of the current instruction being executed .

Execution Order
0
1
2
3
4
…….

Address	Program-Instructions (machine lang.)
00000	01100110 10111101 11001010 11100110
00001	0001010 11010100 10001101 10010100
00010	10011001 11100011 00111001 01000010
00011	10000011 11100011 11001000 11011001
00100	10000100 10000100 0001101 101001010
…….	…….

Assembly..[Assembler]..Machine language



```
li      $t0, 2
li      $t1, 3
add     $t5, $t1, $t0
```

```
01010111
01011110
10010110
11110101
. . . . .
```

Binary bits to be loaded into memory

Assembly program

ASSEMBLER Translator program

Machine Language bits

Assembly programming

Machine Code
0000.0001.0010.1001.0100.0000.0010.000

English
“Put sum of `$t0` and `$t1` into `$t2`”

- Assembly language is a compromise
  - Programmers don’t like machine code
  - Computers can’t understand English

- Mnemonics = symbolic names for instructions

- Labels = named memory locations

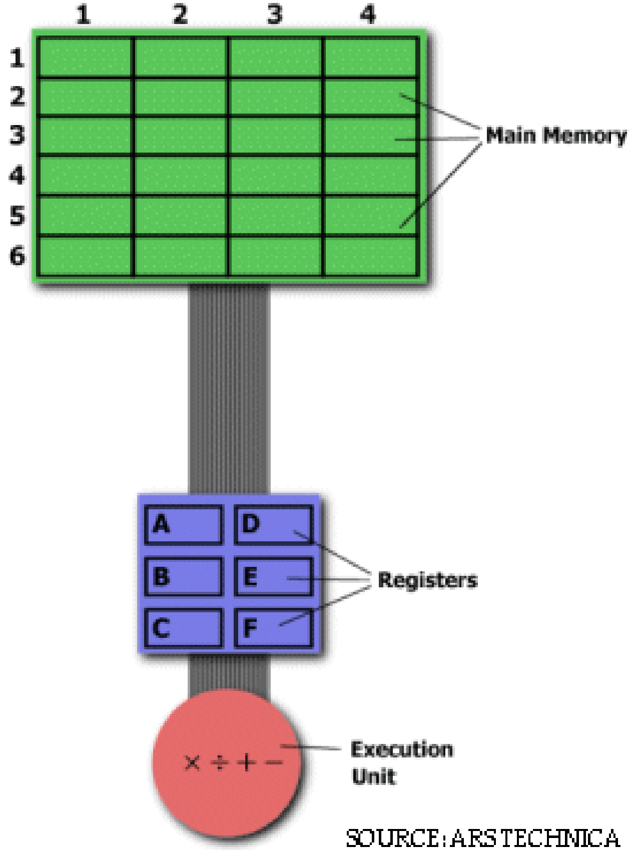
Instruction	Description
`ld $t0, (from)`	Load the word at address from into $t0
`add $t2, $t1, $t0`	Put the sum of $t0 and $t1 into $t2
`st $t2, (there)`	Store the word in $t2 into address there

List of instructions (Assembly Lang.) … Instruction Set

- Arithmetic [Add/Subtract, …]
- Logic [AND, OR, NOT]
- Data [Move, Load, Store, …]
- Control flow [JUMP, …]

Multiply 2 numbers: CISC and RISC

Let's say we want to find the product of two numbers
- one number is stored in location 2:3 and another stored in location 5:2
- then store the product back in the location 2:3.



SOURCE: ARSTECHNICA

CISC

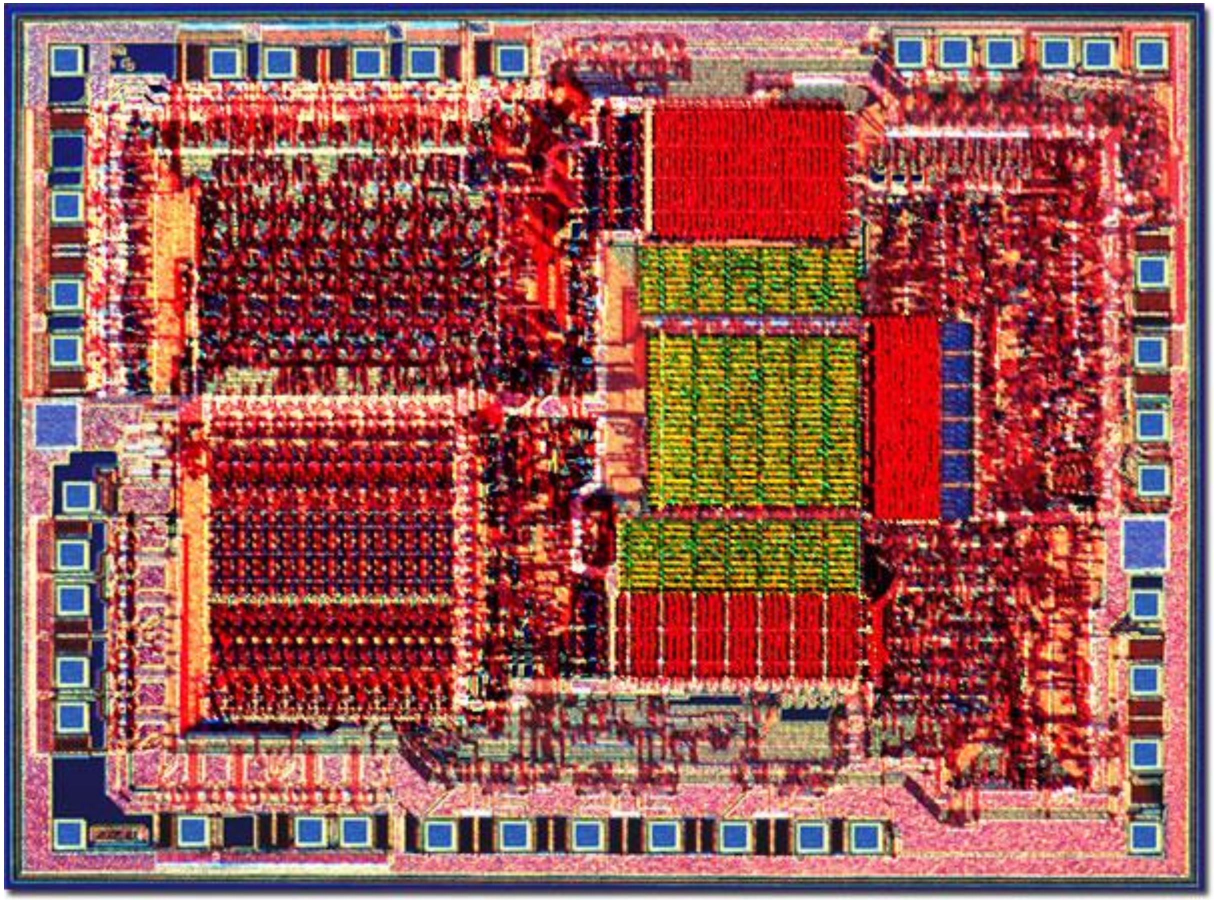
```
MULT 2:3, 5:2
```

RISC

```
LOAD   A, 2:3
LOAD   B, 5:2
PROD   A, B
STORE 2:3, A
```

http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/risccisc/

Intel 8085 (CISC 1970's)



INTEL 8085 (1970’s)

- Single-address architecture: **`ADD B`**
  - ❑ **`(A) = (A) + B`**

- The contents of the operand [register (**`B`**) or memory] are added to the contents of the accumulator and the result is stored in the accumulator.

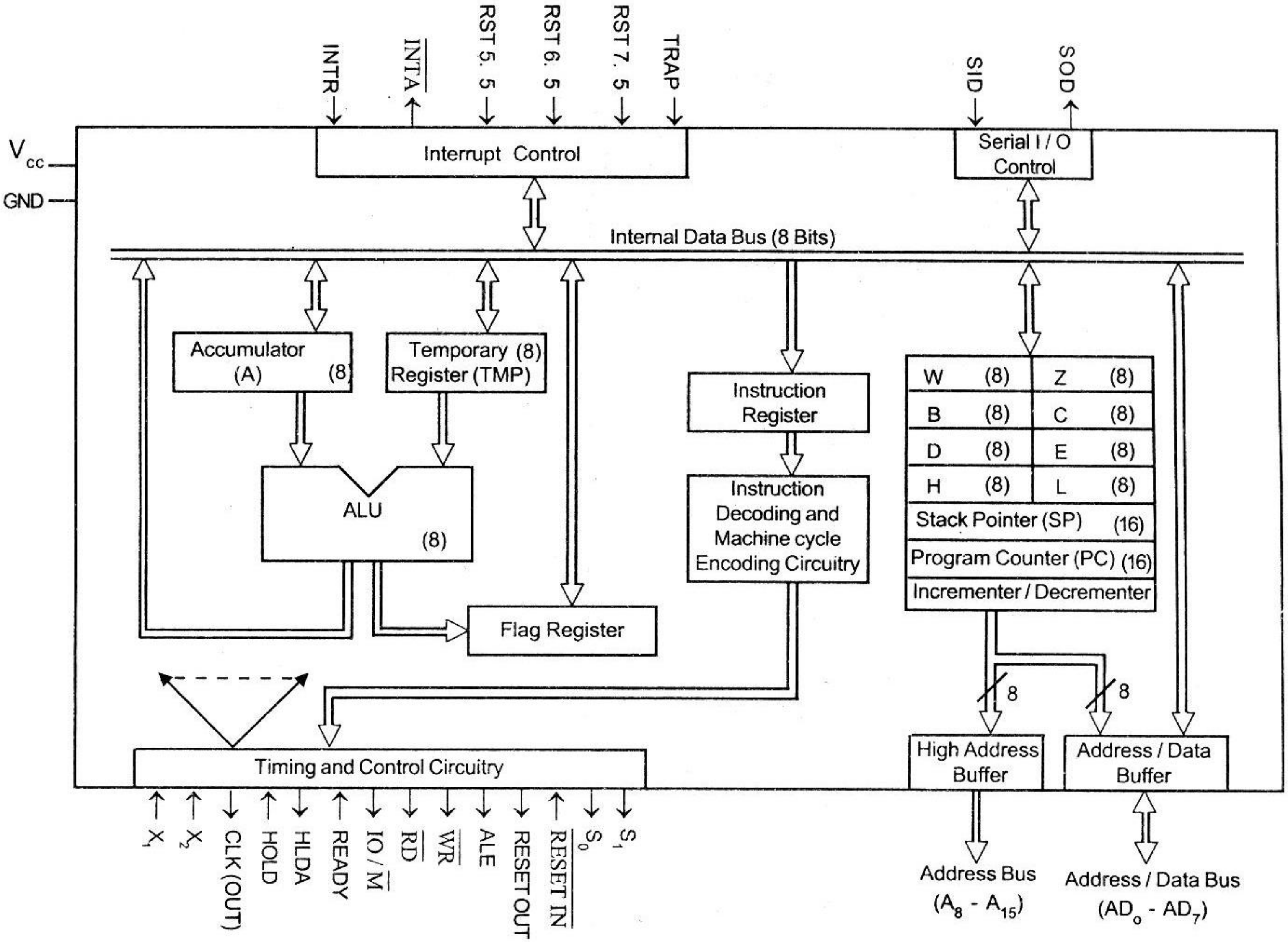
INTEL 8085; Instructions

- The 8085 is an 8-bit processor. It can have up to $2^{\mathbf{8}}$ or 256 instructions
- The 8085 a total of 80 instructions.

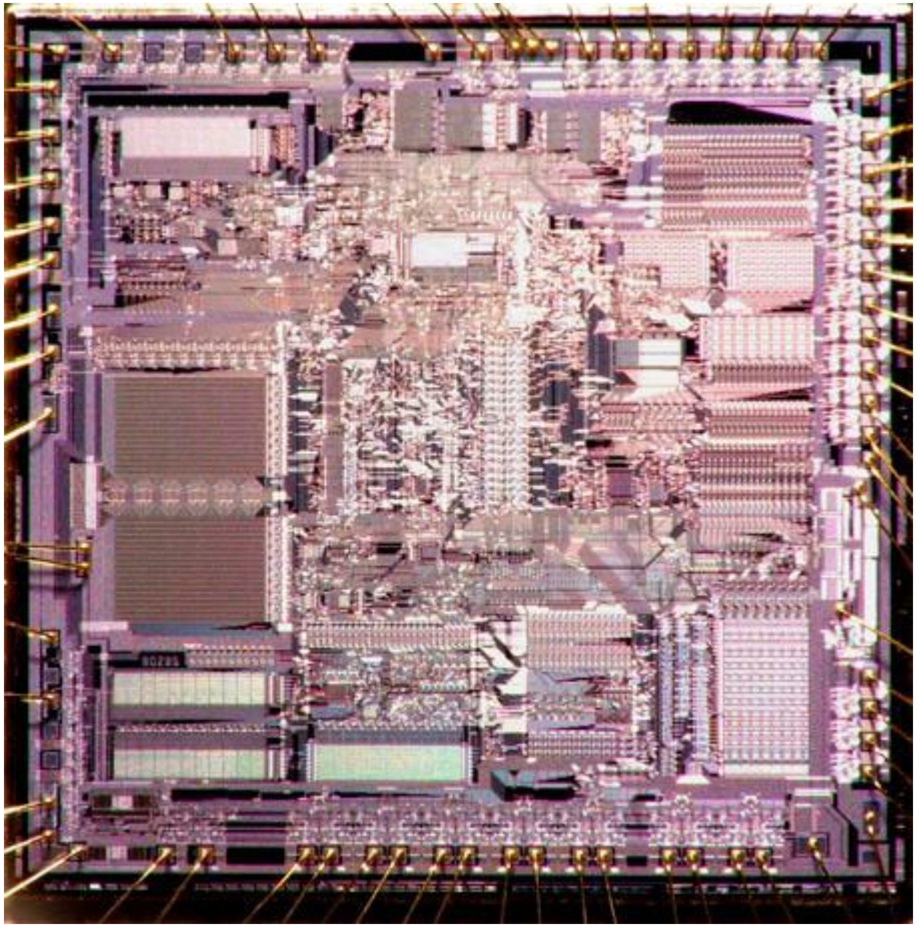
INTEL8085; Instructions classification

1. Data transfer (copy) operations
2. Arithmetic operations
3. Logical operations
4. Branching operations
5. Machine-control operations

INTEL 8085 ARCHITECTURE



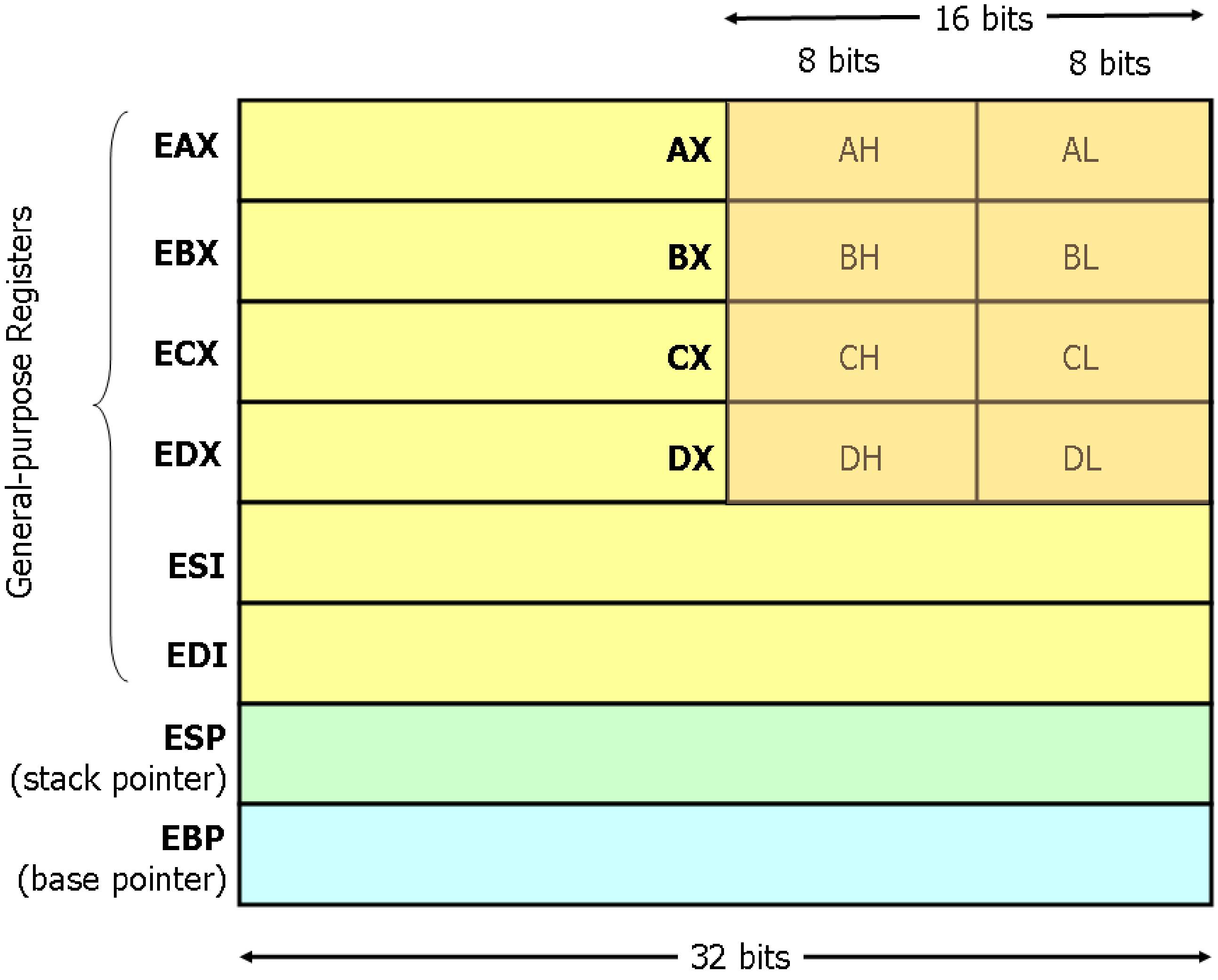
INTEL 80286 (1980’s)



INTEL 80x86 (1980’s)

- Two-address architecture: **`ADD EAX, EBX`**
- Add contents of **`EB`**(base) X register to **`EA`**(accumulator) X register

80x86 Registers



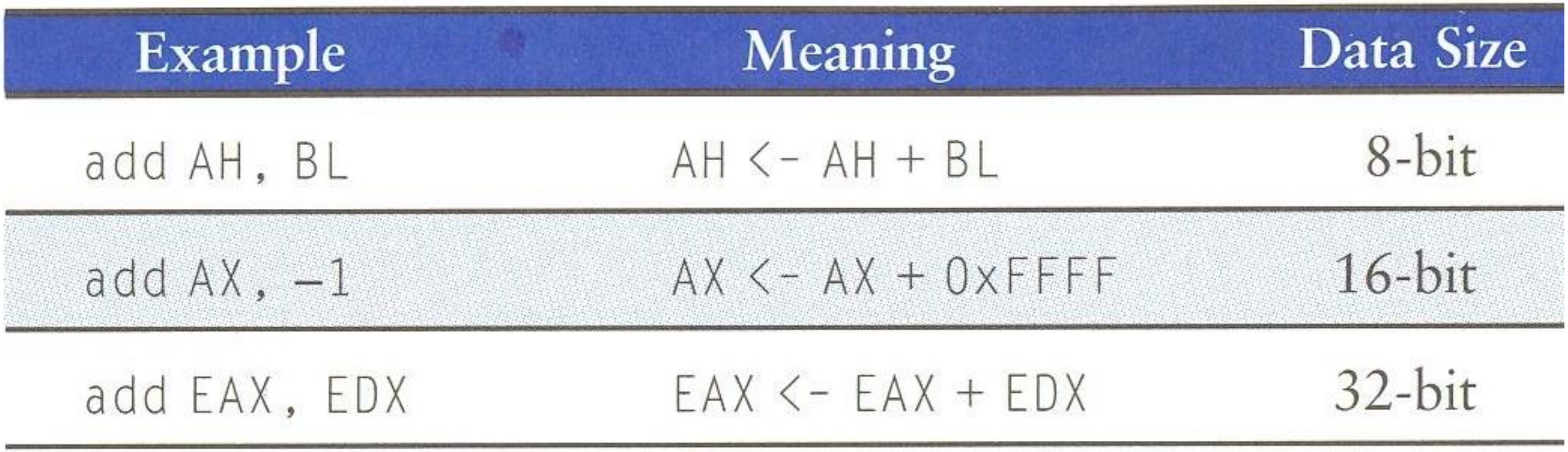
http://www.cs.virginia.edu/~evans/cs216/guides/x86-registers.png

INTEL 80x86

CPU contains a unit called “Register file”. This unit contains the registers of the following types:

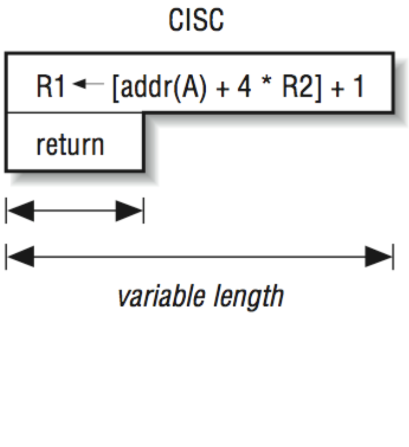
1. *8-bit general registers*:
   AL, BL, CL, DL, AH, BH, CH, DH
2. *16- bit general registers*:
   AX, BX, CX, DX, SP, BP, SI, Dl
3. *32-bit general registers*:
   EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI
   (Accumulator, Base, Counter, Data, Stack pointer, Base pointer, Source index, Destination Index)
4. *Segment registers*:
   ES, CS ,SS, DS, FS, GS
5. *Instruction pointer*:
   EIP

x86 Instructions: 8-; 16-; 32-bit data



Example	Meaning	Data Size
add AH, BL	AH <- AH + BL	8-bit
add AX, –1	AX <- AX + 0xFFFF	16-bit
add EAX, EDX	EAX <- EAX + EDX	32-bit

CISC - RISC



CISC

R1 ← [addr(A) + 4 * R2] + 1

return

variable length

RISC

R3 ← 4 * R2

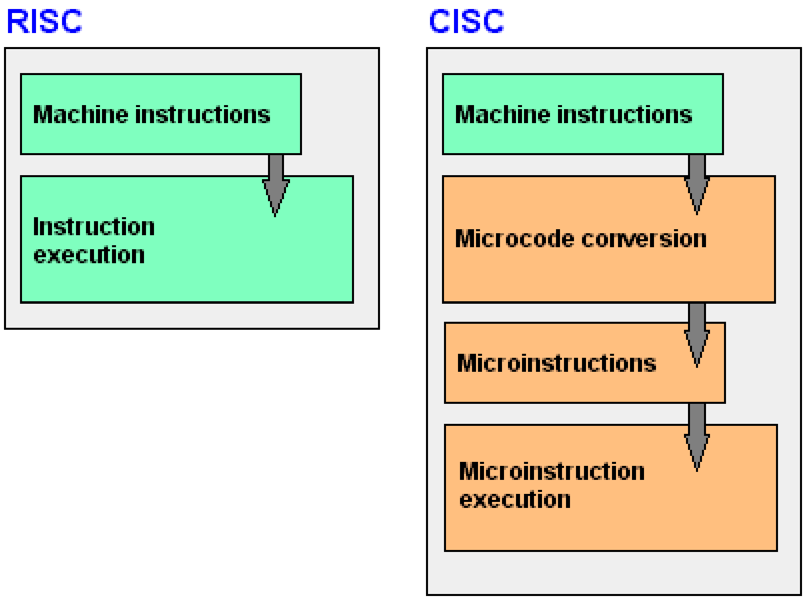
R3 ← addr (A) + R3

R5 ← [R4]

R1 ← R5 + 1

return

fixed length



RISC

Machine instructions

Instruction execution

CISC

Machine instructions

Microcode conversion

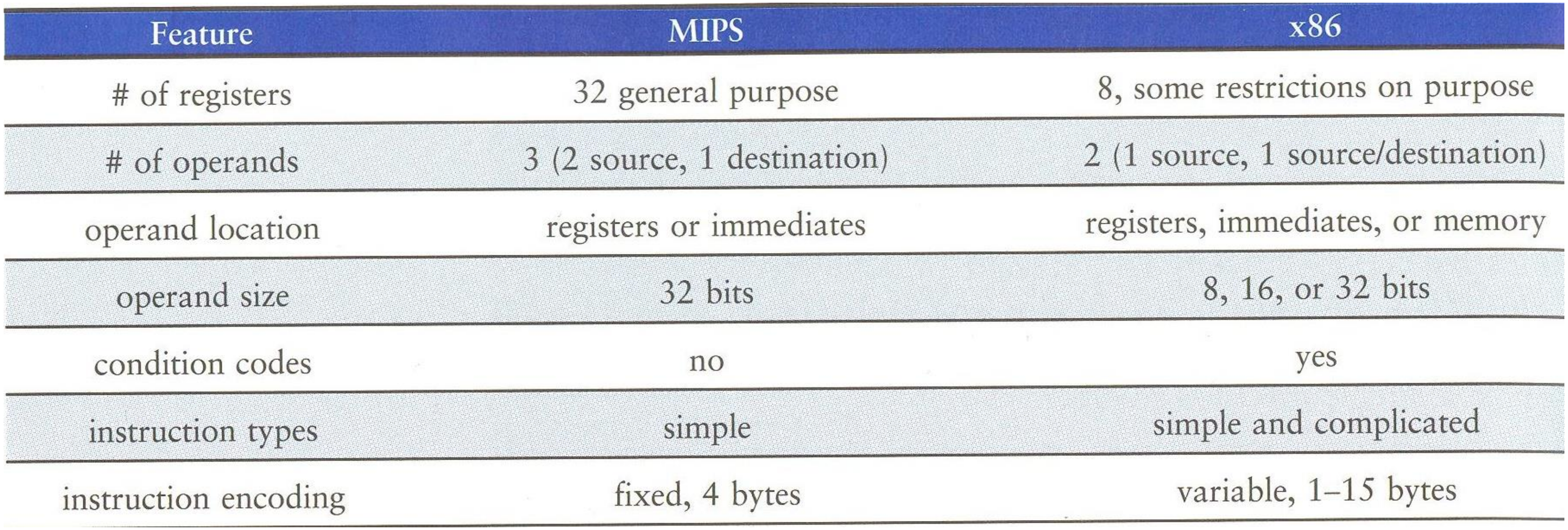
Microinstructions

Microinstruction execution

http://cnx.org/

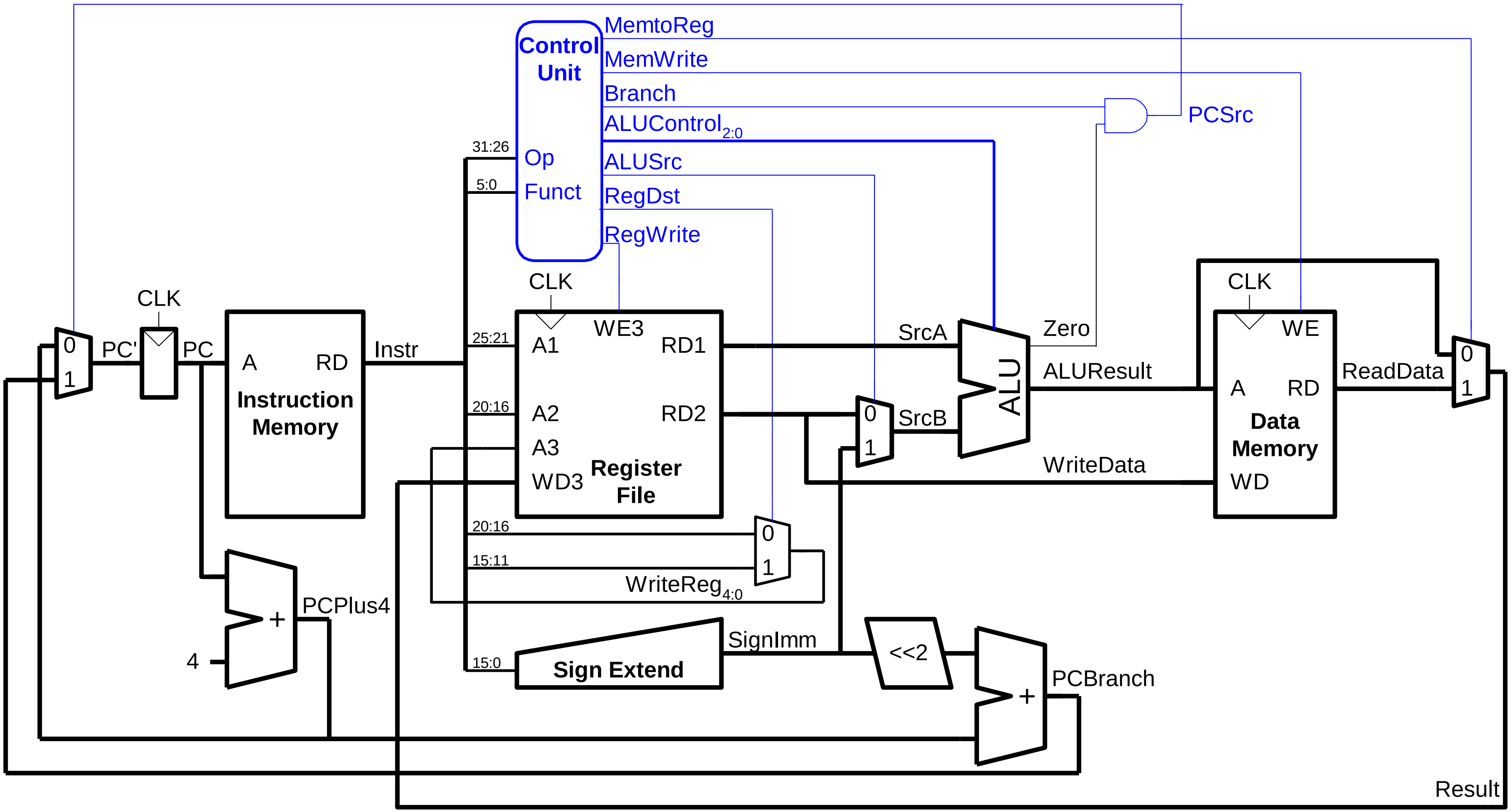
http://www.pcmag.com

MIPS vs x86



Feature	MIPS	x86
# of registers	32 general purpose	8, some restrictions on purpose
# of operands	3 (2 source, 1 destination)	2 (1 source, 1 source/destination)
operand location	registers or immediates	registers, immediates, or memory
operand size	32 bits	8, 16, or 32 bits
condition codes	no	yes
instruction types	simple	simple and complicated
instruction encoding	fixed, 4 bytes	variable, 1–15 bytes

MIPS (Single Cycle MIPS CPU)



MIPS

- Three-address architecture: `add $t1, $t2, $t3`
- `($t1) = ($t2) + ($t3)`

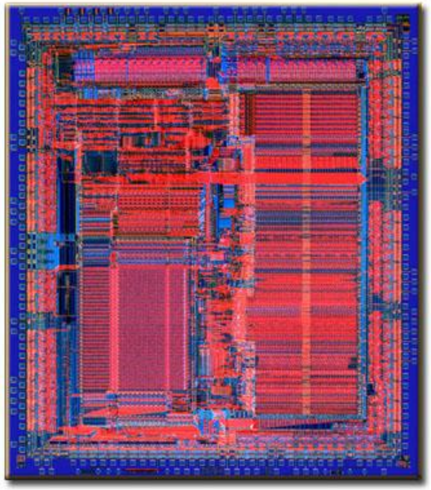
- `$t1,$t2,$t3` = Names of 3 registers

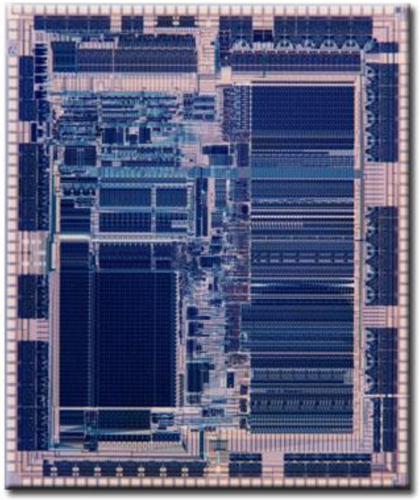
MIPS Register Set

Software
http://courses.missouristate.edu/kenvollmar/mars/

Name	Register Number	Usage
`$0`	0	the constant value 0
`$at`	1	assembler temporary
`$v0-$v1`	2-3	Function return values
`$a0-$a3`	4-7	Function arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved variables
`$t8-$t9`	24-25	more temporaries
`$k0-$k1`	26-27	OS temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	Function return address

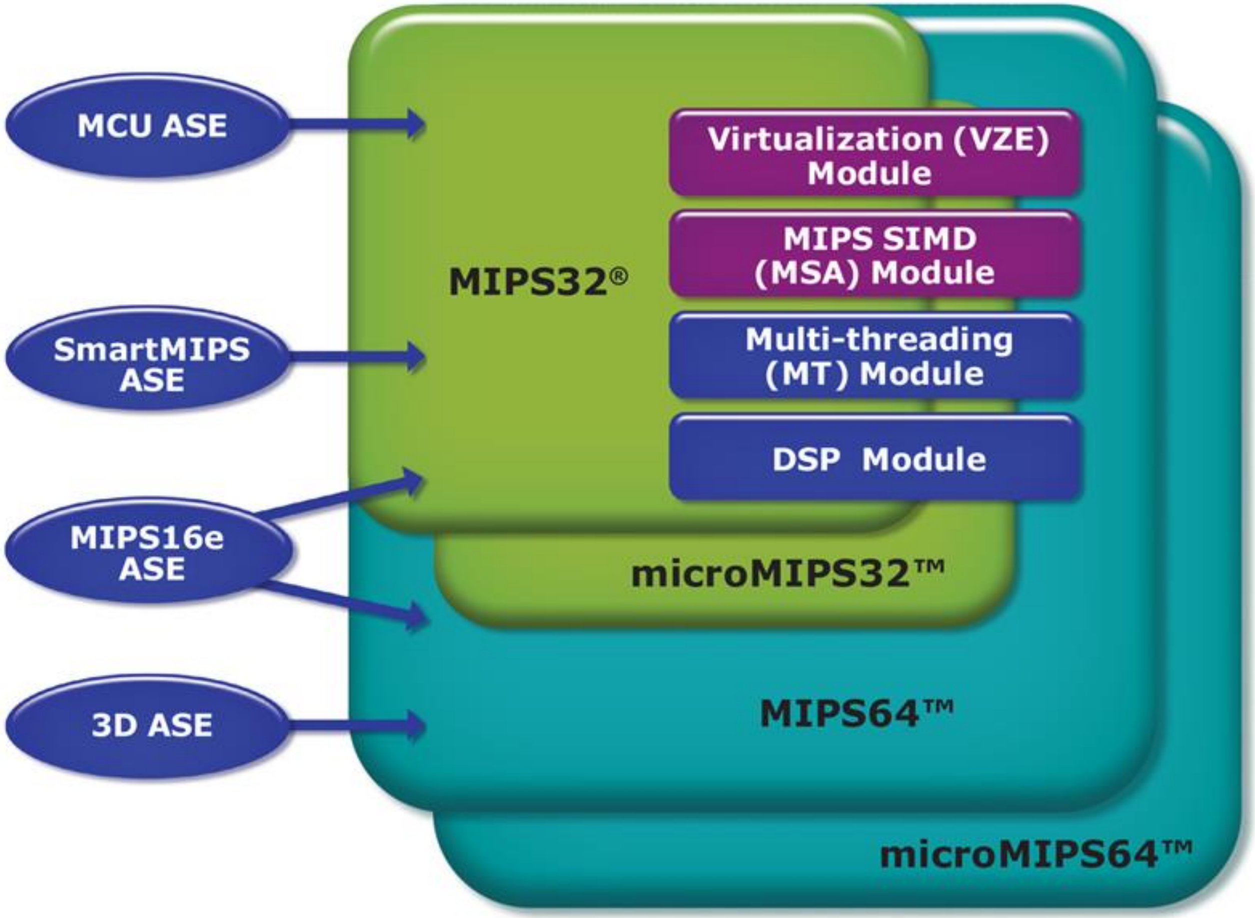
MIPS2000 RISC Processor; The start





1986

Today …


MCU ASE
SmartMIPS ASE
MIPS16e ASE
3D ASE
MIPS32®
Virtualization (VZE) Module
MIPS SIMD (MSA) Module
Multi-threading (MT) Module
DSP Module
microMIPS32™
MIPS64™
microMIPS64™

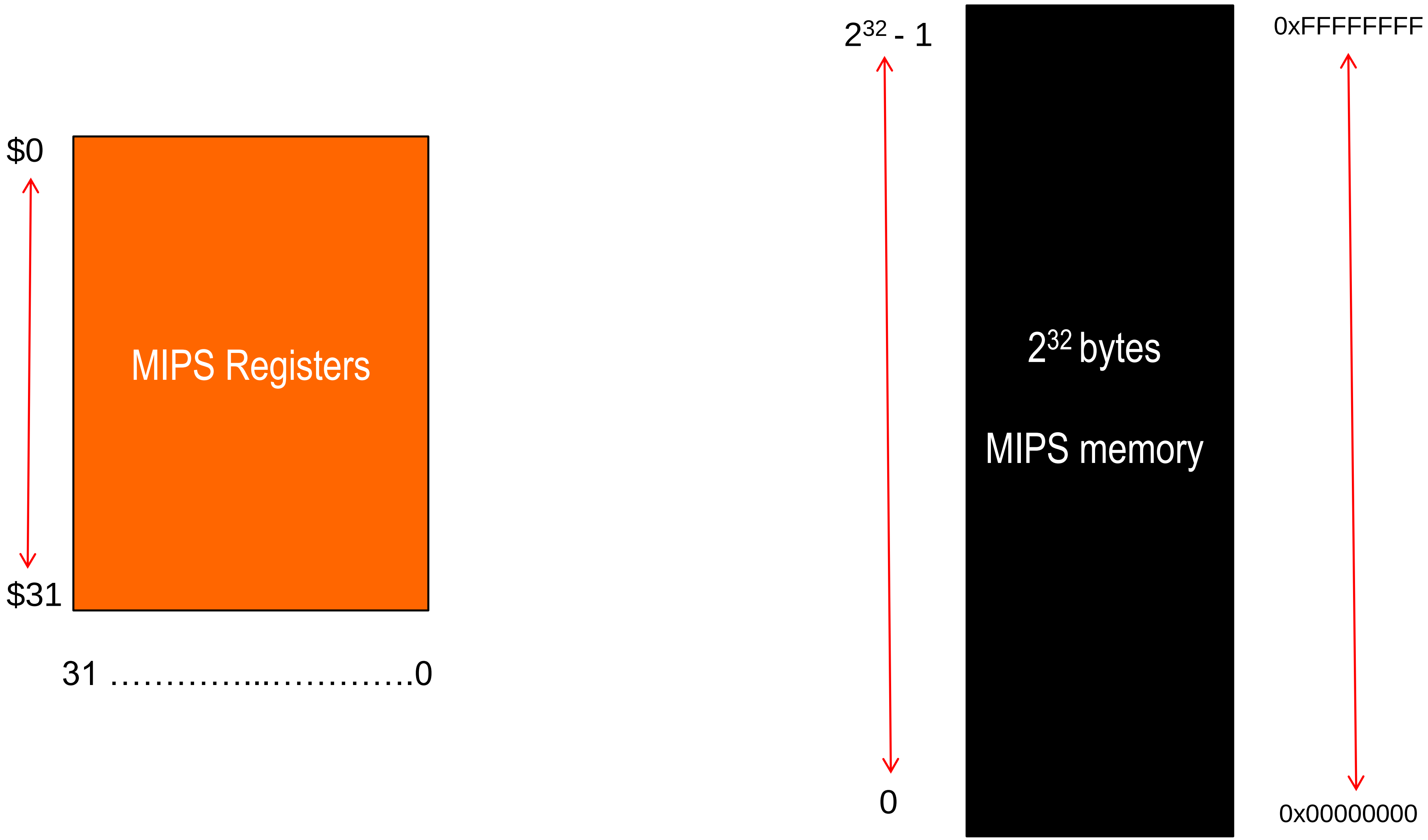
MIPS is a [Load/Store Architecture]

- Load (from memory to CPU registers)
- Store (from CPU registers to memory).

Registers, Data transfers

- In a CPU, registers store temporary data
  - Relatively slow Operations/Transfers: To-and-from memory (Load/Store)
  - Fast Operations/Transfers: Between registers.

MIPS R2000 RISC Processor



$0

MIPS Registers

$31

31 ……………..…………..0

$2^{32} - 1$

2^{32} bytes

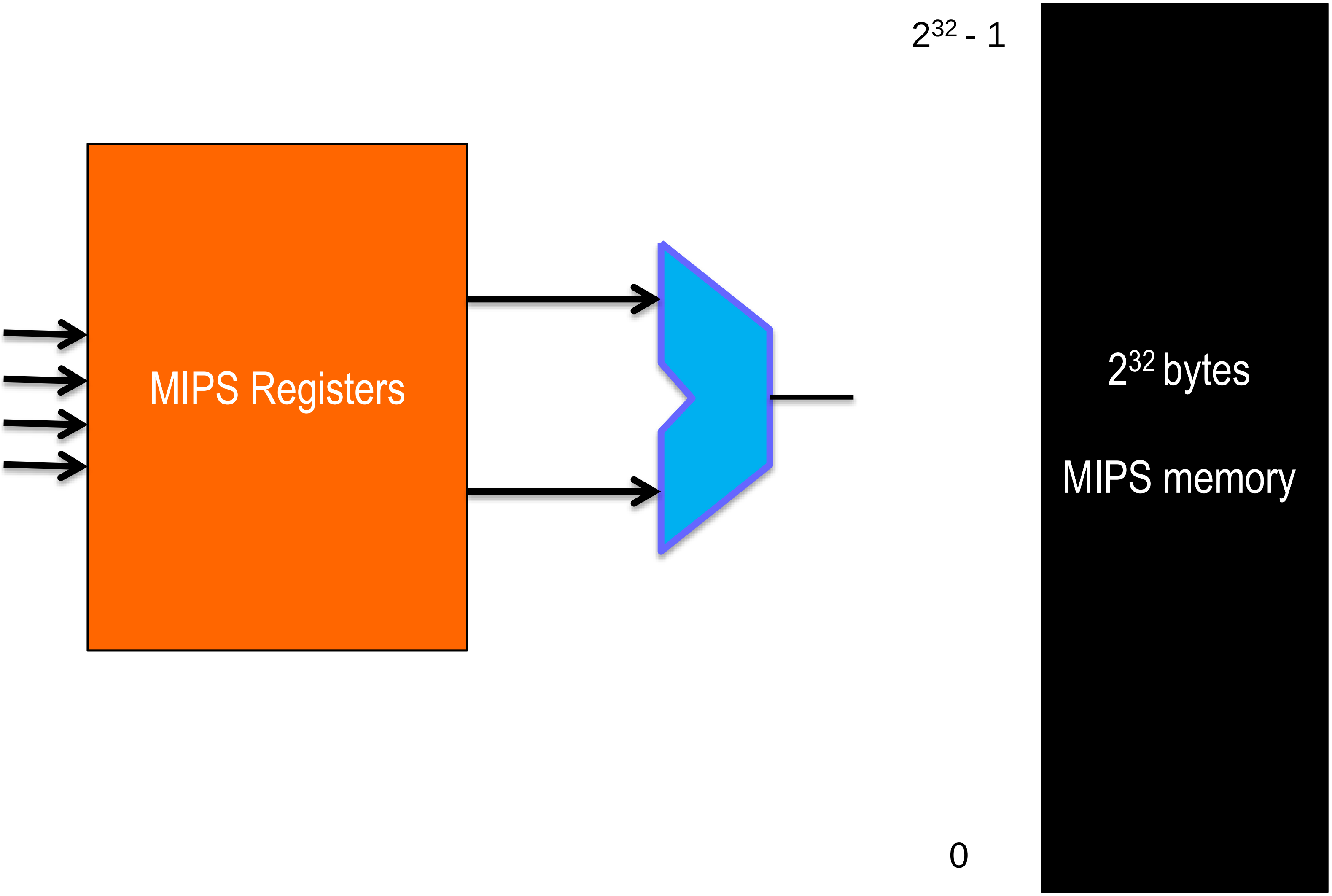
MIPS memory

0

0xFFFFFFFF

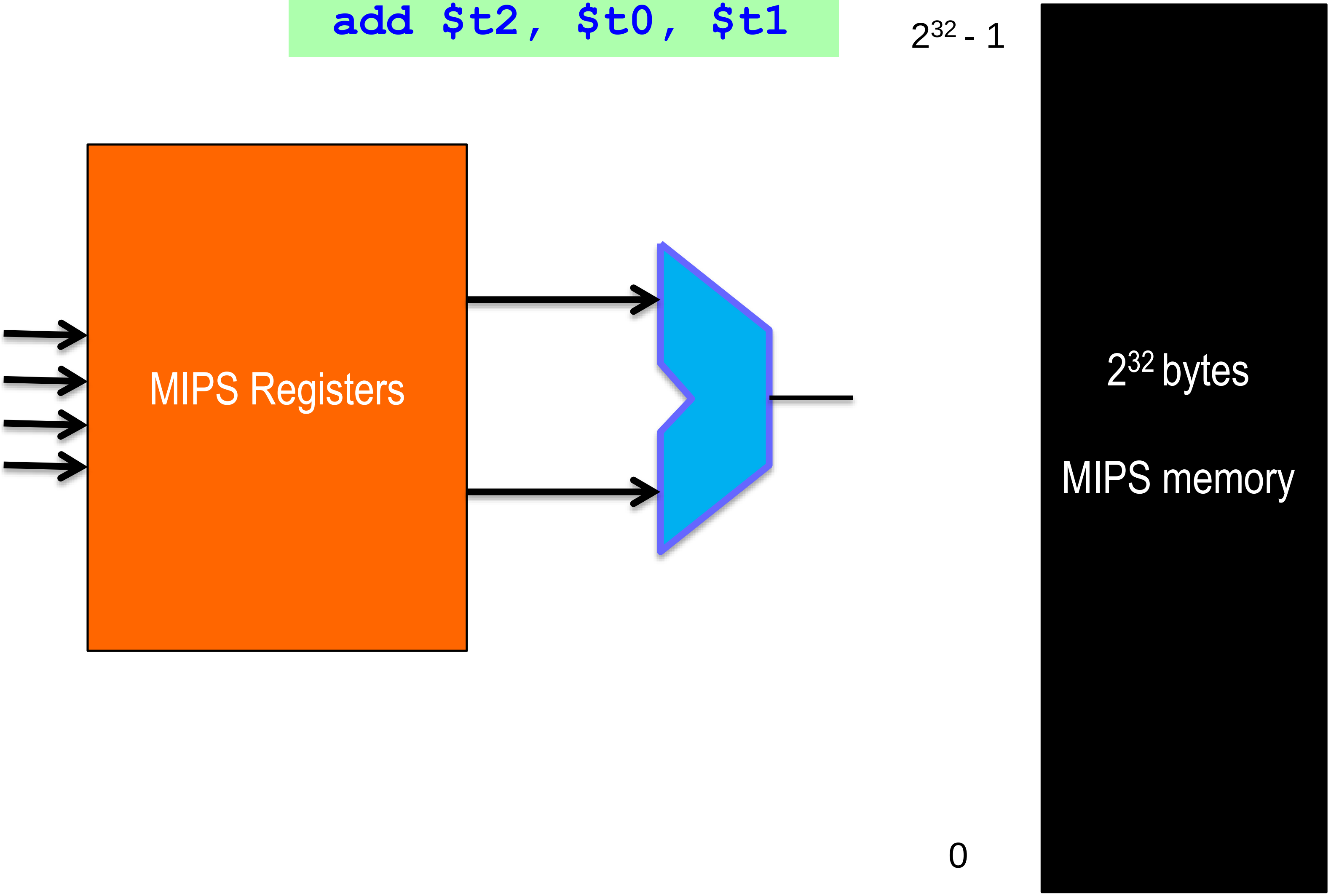
0x00000000

MIPS R2000 RISC Processor



MIPS Instruction

```
add $t2, $t0, $t1
```



MIPS Registers

2^{32} - 1

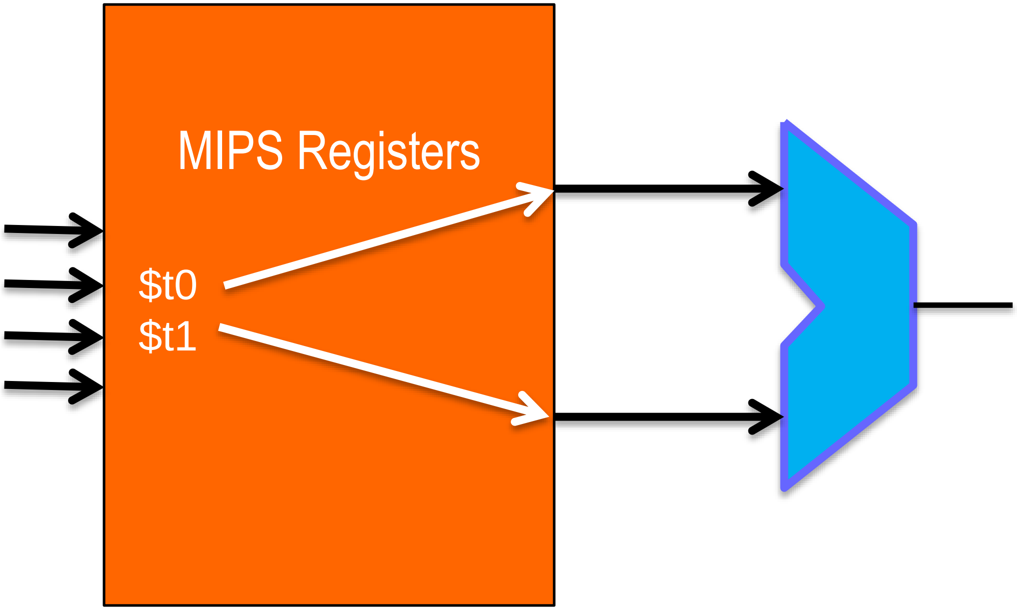
2^{32} bytes

MIPS memory

0

MIPS Instruction

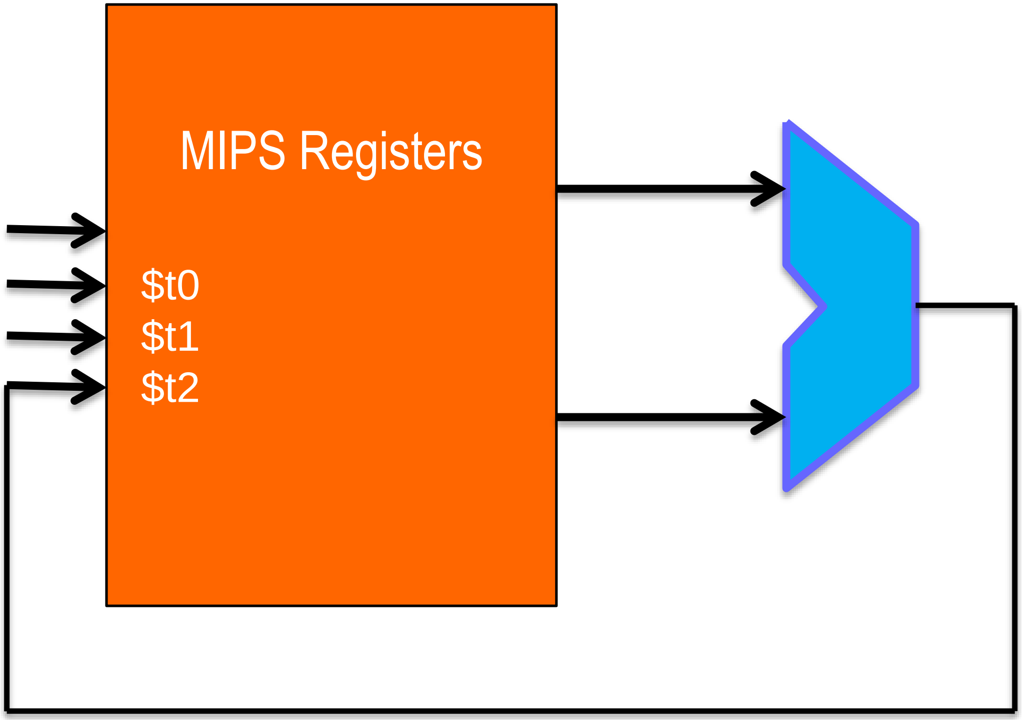
`add $t2, $t0, $t1`



Registers: `$t2, $t0, $t1`

MIPS Instruction

`add $t2, $t0, $t1`



Registers: `$t2, $t0, $t1`

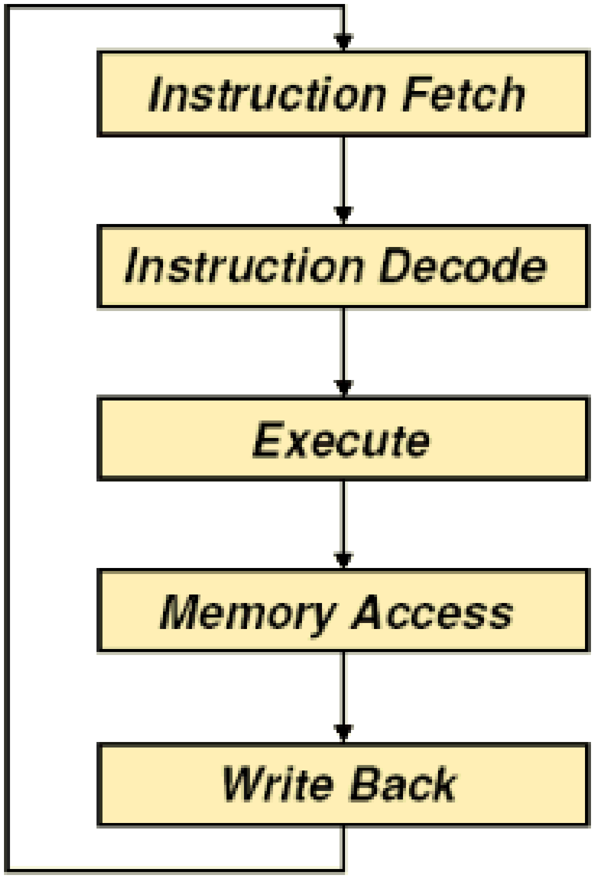
MIPS Register Set

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address

Operands: Registers

- Registers:
  - `$` before name
  - Example: `$0`, “register zero”, “dollar zero”
- Registers used for specific purposes:
  - `$0` always holds the constant value 0.
  - the *saved registers*, `$s0-$s7`, used to hold variables
  - the *temporary registers*, `$t0-$t9`, used to hold intermediate values during a larger computation
  - ….

Fetch-Decode-Execute-Memory-WriteBack



For each instruction

The explanation of the 5 cycle

Fetch-Decode-Execute-Memory-WriteBack

- **IF: Instruction Fetch**
  - The next instruction is read from the program memory
- **ID: Instruction Decode**
  - The instruction operand is decoded, and the registers needed are read.
- **EX: Execute**
  - Executes the instruction and performs calculations with the ALU (arithmetic and logic unit)
- **MEM: Memory Access**
  - Reads or writes to the data memory, such as with the lw (Load Word) and sw (Store Word) instructions
- **WB: Write Back**
  - Write back the results in the resulting register

`add $t0, $s1, $s2`

Assembly Language

- **IF**: get the (below) instruction `add` from memory

op code	rs ($s1)	rt ($s2)	rd ($t0)	shamt	function
000000	00011=3	00100=4	00111=7	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

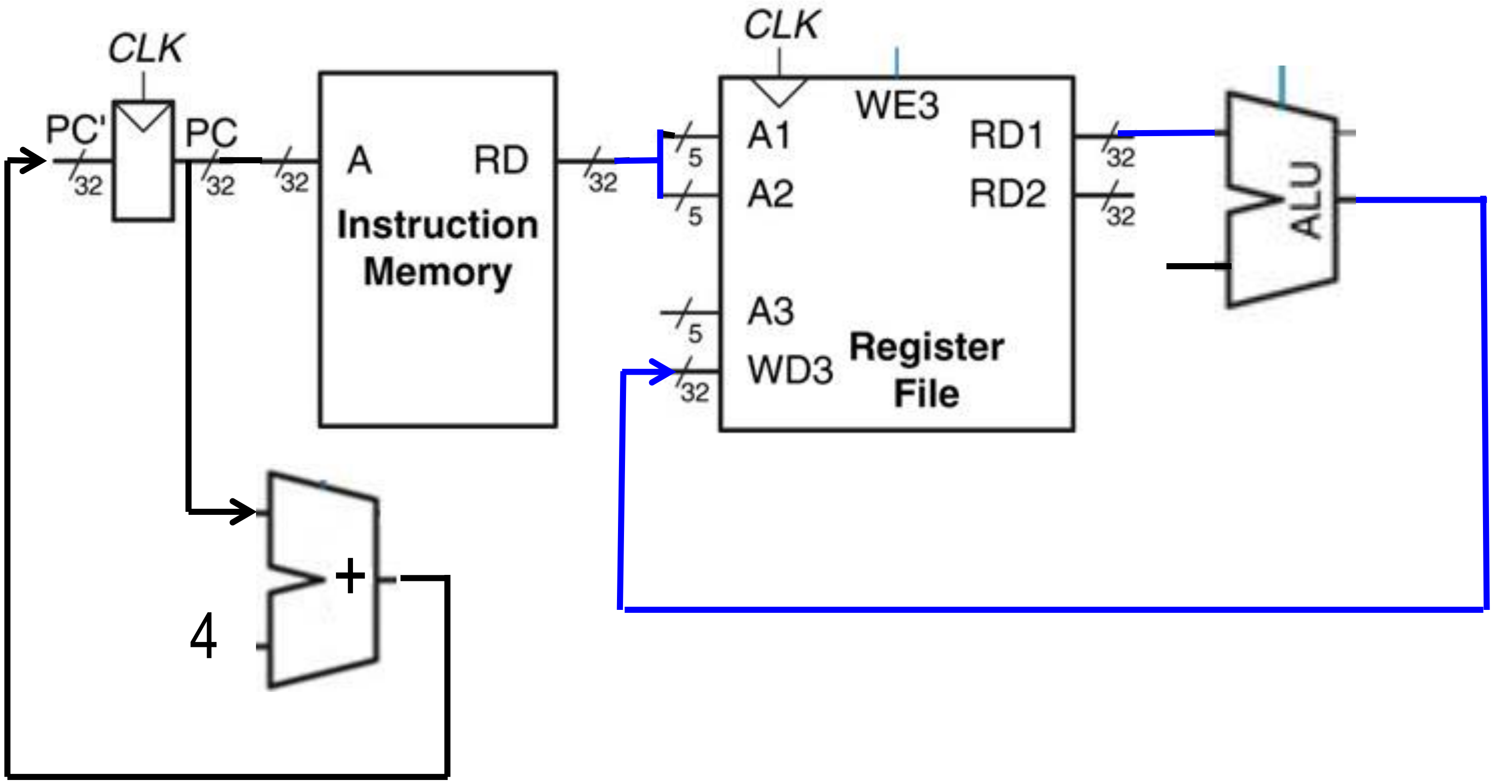
Machine Language

- **ID**: instruction function/read registers
  - R-instruction; > Op code = 000000
  - `add` instruction because: 100000
  - Reg's source: `$s1, $s2`; contents: `$s1` = 3, `$s2` = 4
- **EX**: `add` 3 + 4 = 7 (00111)
- **MEM**: No Operation
- **WB**: The result back to reg. `$t0` = 00111 = 7

Explanations

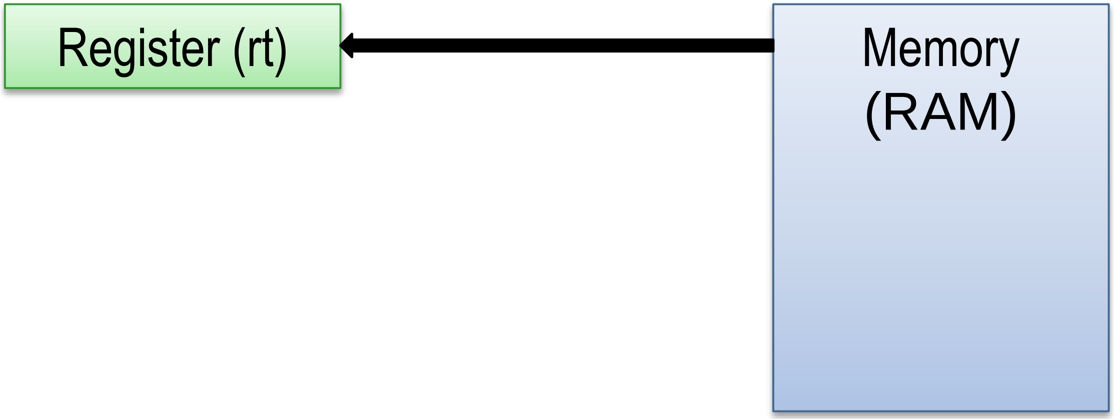
- **op** (**op**eration code)
- **rs** (First source **r**egister operand)
- **rt** (Second source **r**egister operand)
- **rd** (**D**estination **r**egister operand)
- **shamt** (**Sh**ift **am**ount - used in shift instructions)
- **funct** (Select the variant of the operation in the op code field)

add $t0, $s1, $s2


CLK
PC'
32
PC
32
32
A
RD
Instruction Memory
32
CLK
WE3
A1
A2
A3
WD3
5
5
5
32
RD1
RD2
32
32
Register File
ALU
+
4

`lw $rt, offset($rs)`

Read (Copy) a word from memory (`offset($rs)`) into register (rt)



Indexed (Based) Addressing

`lw $t2, 32`$_{10}$`($t0)`

Read from memory (word 32_{10}) into $t2

1. **F**: Get the instruction `lw` (load word) from memory

op code	rs ($t0)	rt ($t2)	Address/immediate
100011	00000	01010	0000 0000 0010 0000
6	5	5	16

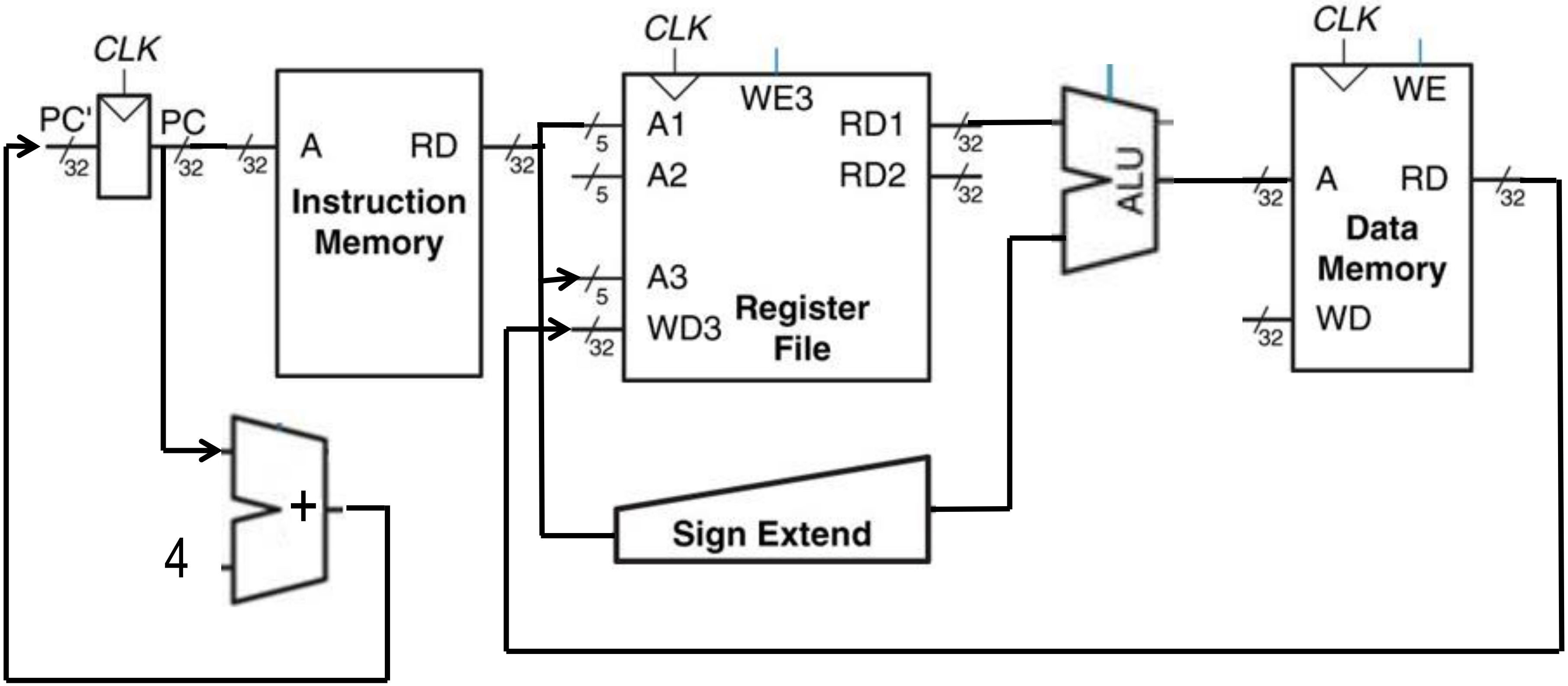
2. **D**: The op code > 010111
   - 010111 is `lw`
   - get contents of `$t0`; (`$t0` = 0)

3. **EX**: Add 32_{10} to memory location: (use Sign Extend)
   `$t0` +32_{10} = 32_{10}= 00000000000000000000000000100000 = 0x0020

4. **MEM**: Load the word stored at: 0x0020

5. **WB**: Store loaded value to `$t2`

lw $t2, 32₁₀($t0)


CLK
PC'
PC
32
32
32
32
A
RD
Instruction
Memory
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
5
5
5
32
Register
File
32
32
ALU
CLK
WE
A
RD
Data
Memory
WD
32
32
32
+
4
Sign Extend

MIPS Example-1;

Expression `a = b + c;`

MIPS code: `add $t0, $s1, $s2`

MIPS Example-2;

Expressions:

```
a = b + c + d;
e = f - a;
```

MIPS code:

```
add $t0, $s1, $s2    >  c + d
add $s0, $t0, $s3    >  b + (c + d)
sub $s4, $s5, $s0    >  f – [ b + (c + d)]
```

MIPS Example-3;

Expression : `a = (b + c) - (d + e);`

MIPS code:

```
add $t0, $s1, $s2   >   (b+c)
add $t1, $s3, $s4   >   (d+e)
sub $s0, $t0, $t1
```

MIPS Example-4

Expression: **a = b + c + d - e**

```
    add $t0, $s1, $s2    >  c + d
    add $t0, $t0, $s3    >  b + (c+d)
    sub $s0, $t0, $s4    > [b + (c+d)]- e
```

Add immediate

```
a = b + 10;

addi $s0,$s1,10    (in MIPS)
```

```
a = b - 10;

addi $s0,$s1,-10 (in MIPS)
```

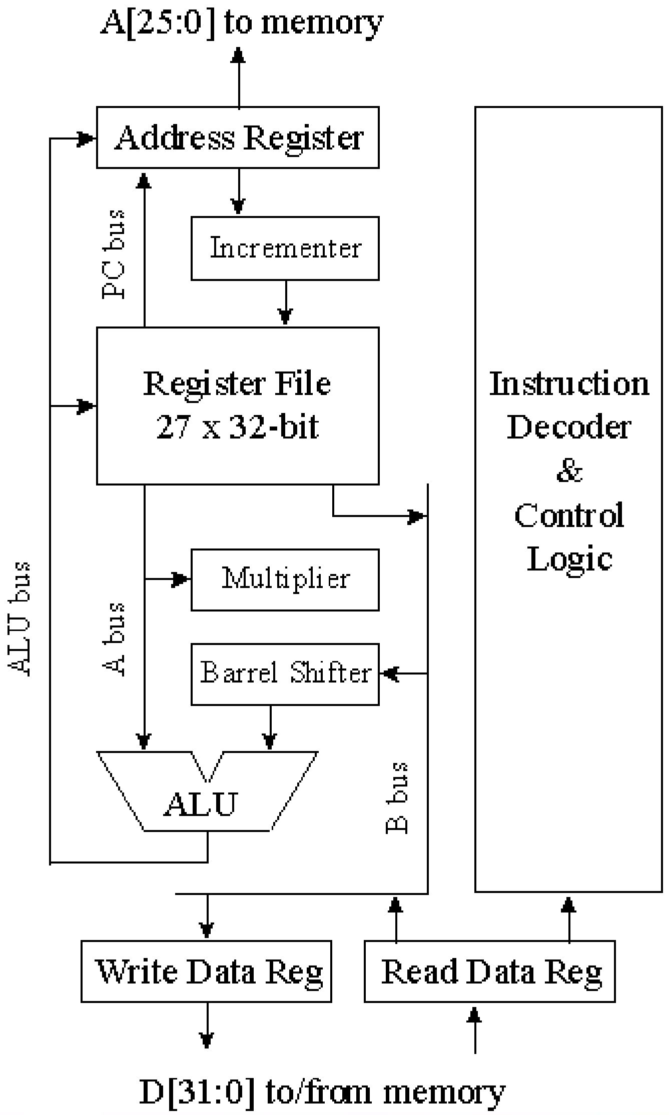
Acorn RISC Machine: ARM

- The ARM is a Reduced Instruction Set Computer (RISC)

    - Register file: The processor has a total of 37 registers made up of 31 general 32-bit registers and 6 status registers
    - Booth Multiplier
    - Barrel shifter
    - Arithmetic Logic Unit (**ALU**)
    - Control Unit.

http://www.arm.com/products/CPUs/index.html

ARM2 (1996) Datapath



http://homepages.thm.de/~hg10013/Lehre/MMS/WS0304_SS04/Ioannis/text.htm

ARM7 (2000) DataPath

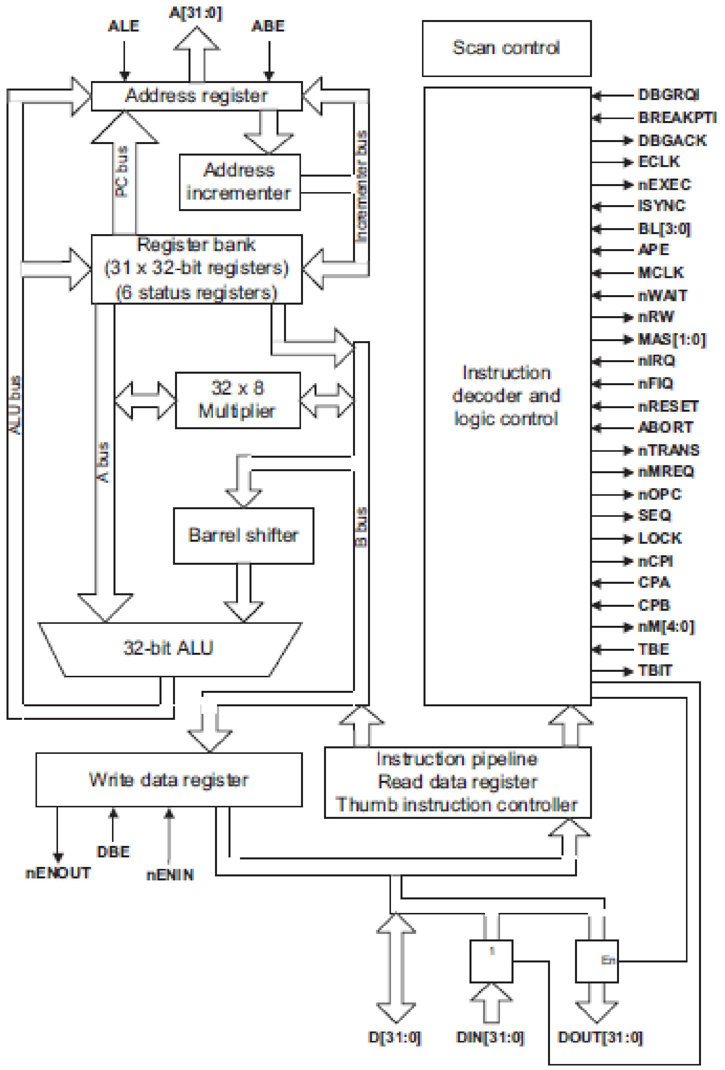
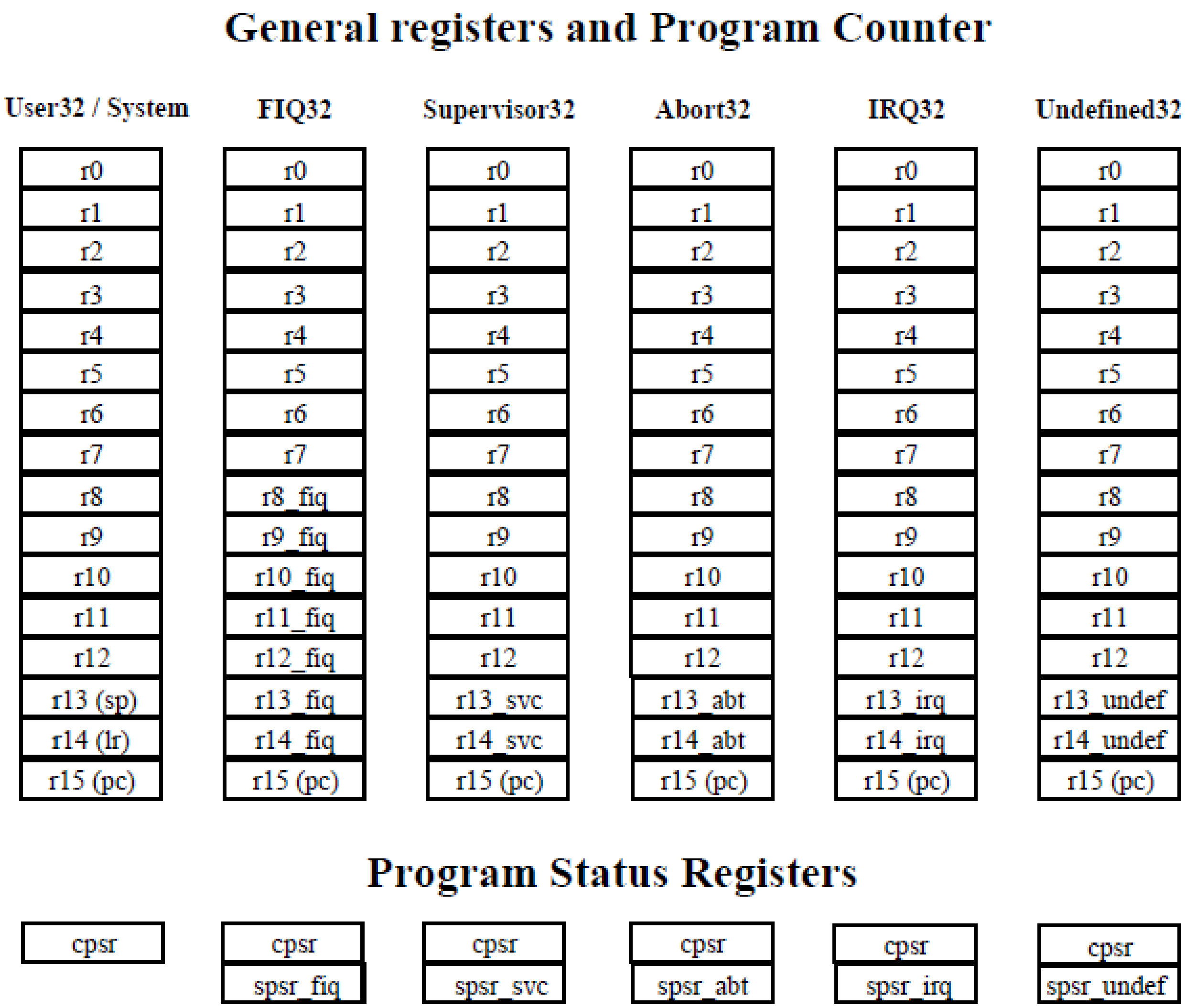


Figure 1 ARM7TDMI core diagram

General Registers



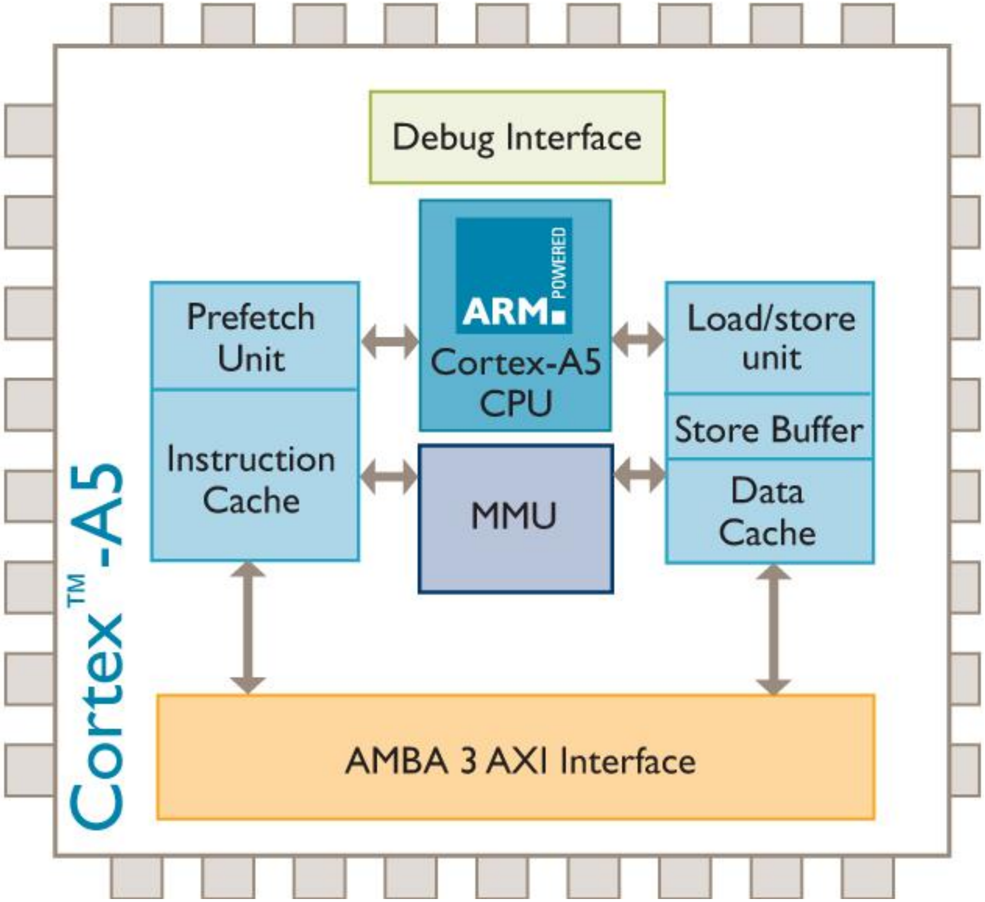
General registers and Program Counter

User32 / System	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13 (sp)	r13_fiq	r13_svc	r13_abt	r13_irq	r13_undef
r14 (lr)	r14_fiq	r14_svc	r14_abt	r14_irq	r14_undef
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)

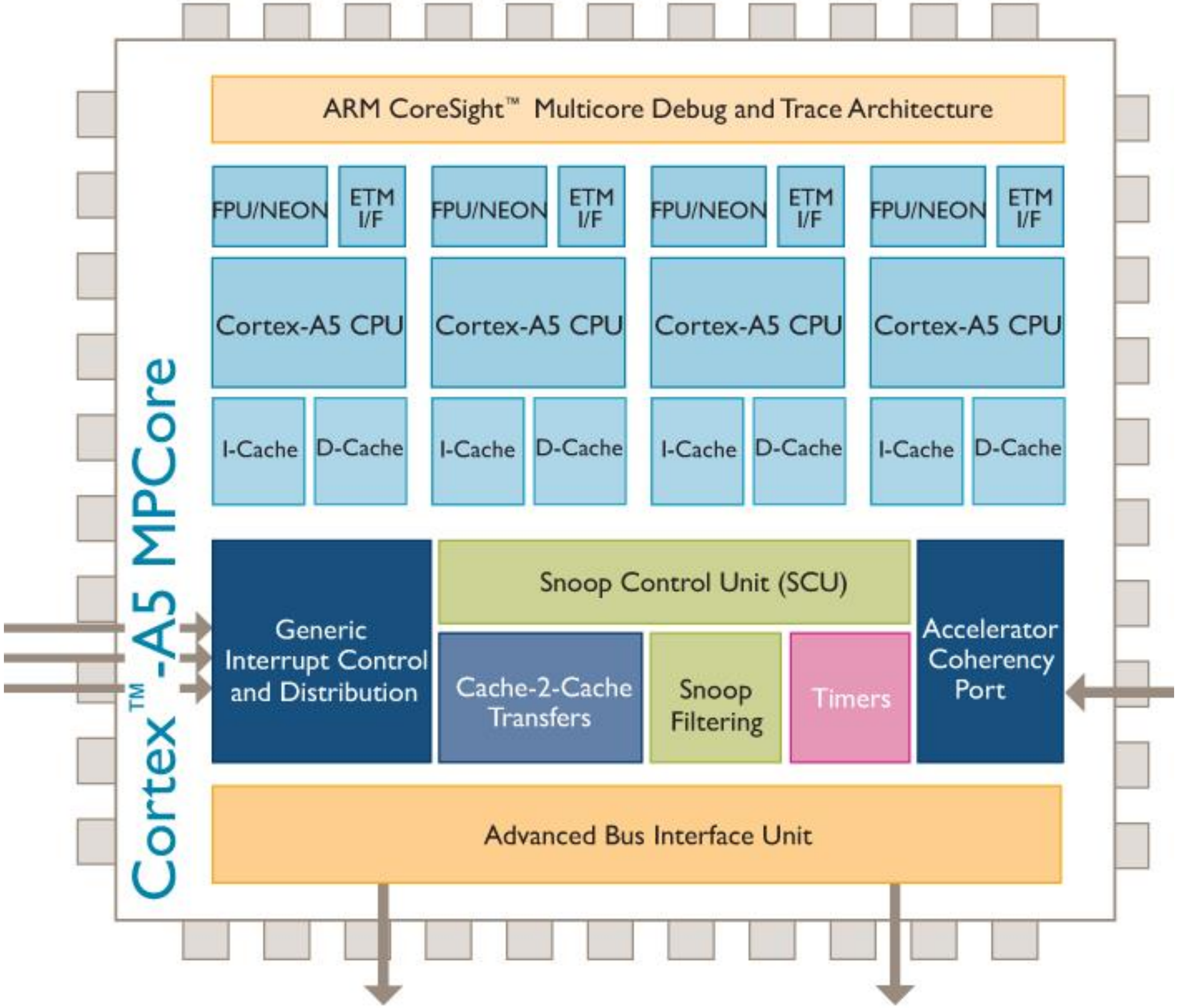
Program Status Registers

cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	spsr_fiq	spsr_svc	spsr_abt	spsr_irq	spsr_undef

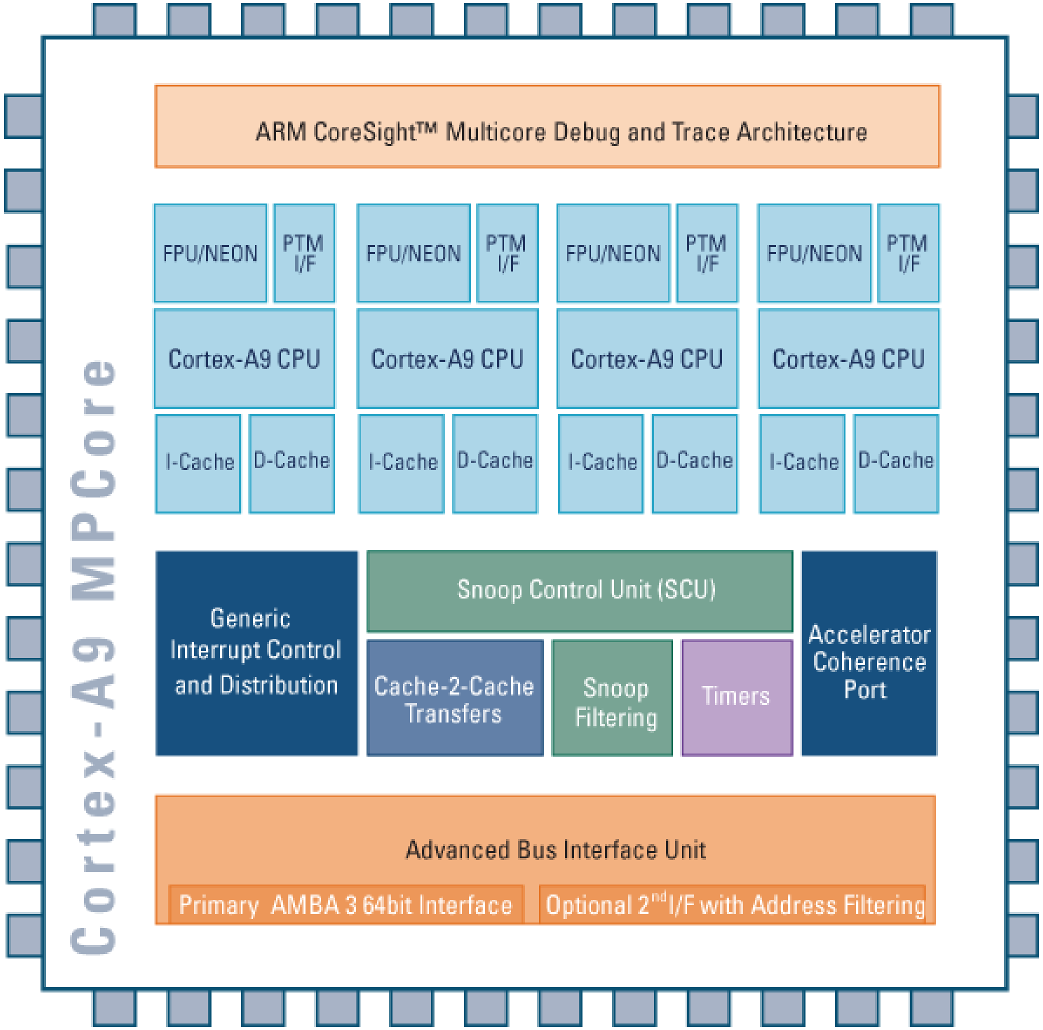
ARM Cortex-5


Debug Interface
ARM POWERED
Cortex-A5 CPU
Prefetch Unit
Instruction Cache
MMU
Load/store unit
Store Buffer
Data Cache
AMBA 3 AXI Interface
Cortex™-A5

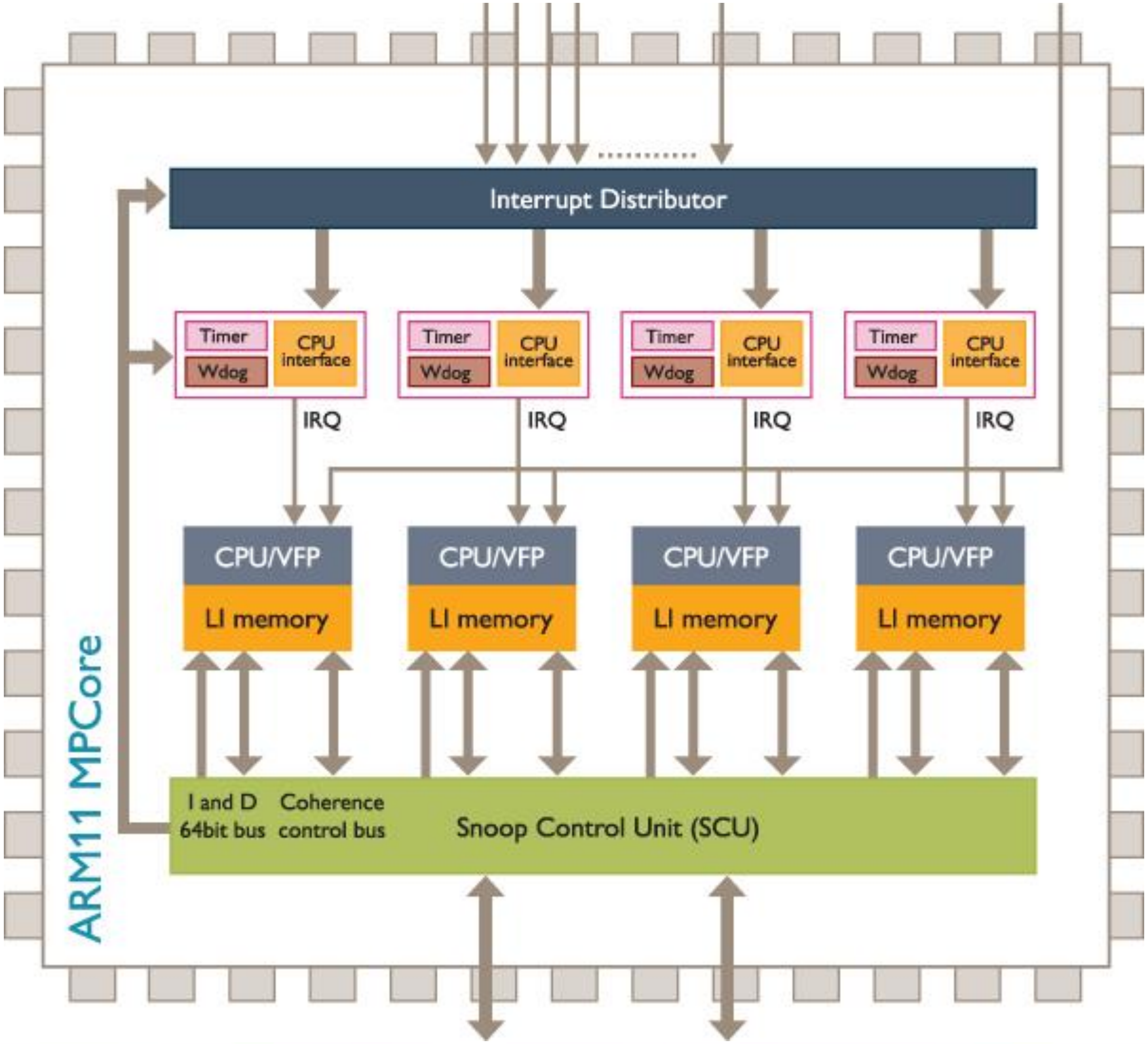
ARM Cortex-A5 MPCore



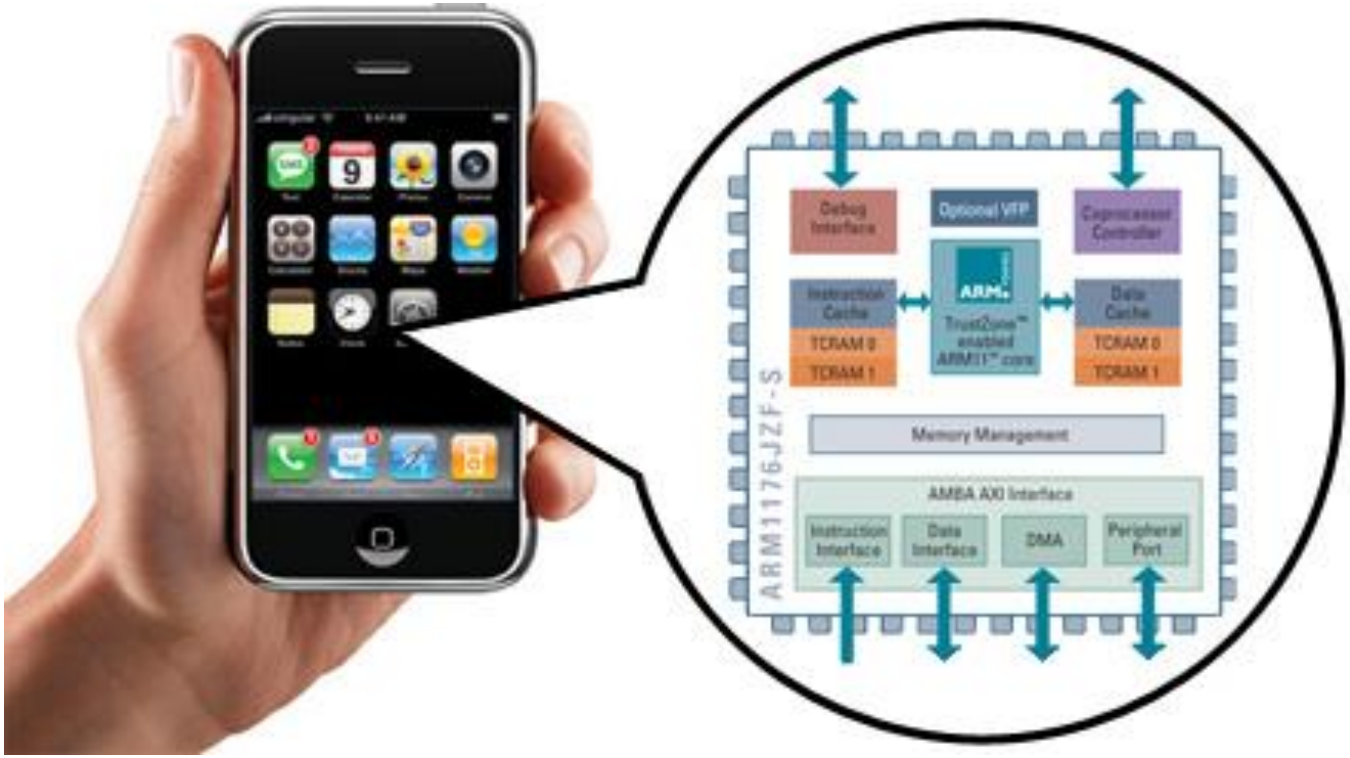
ARM-Cortex A9 MPCore



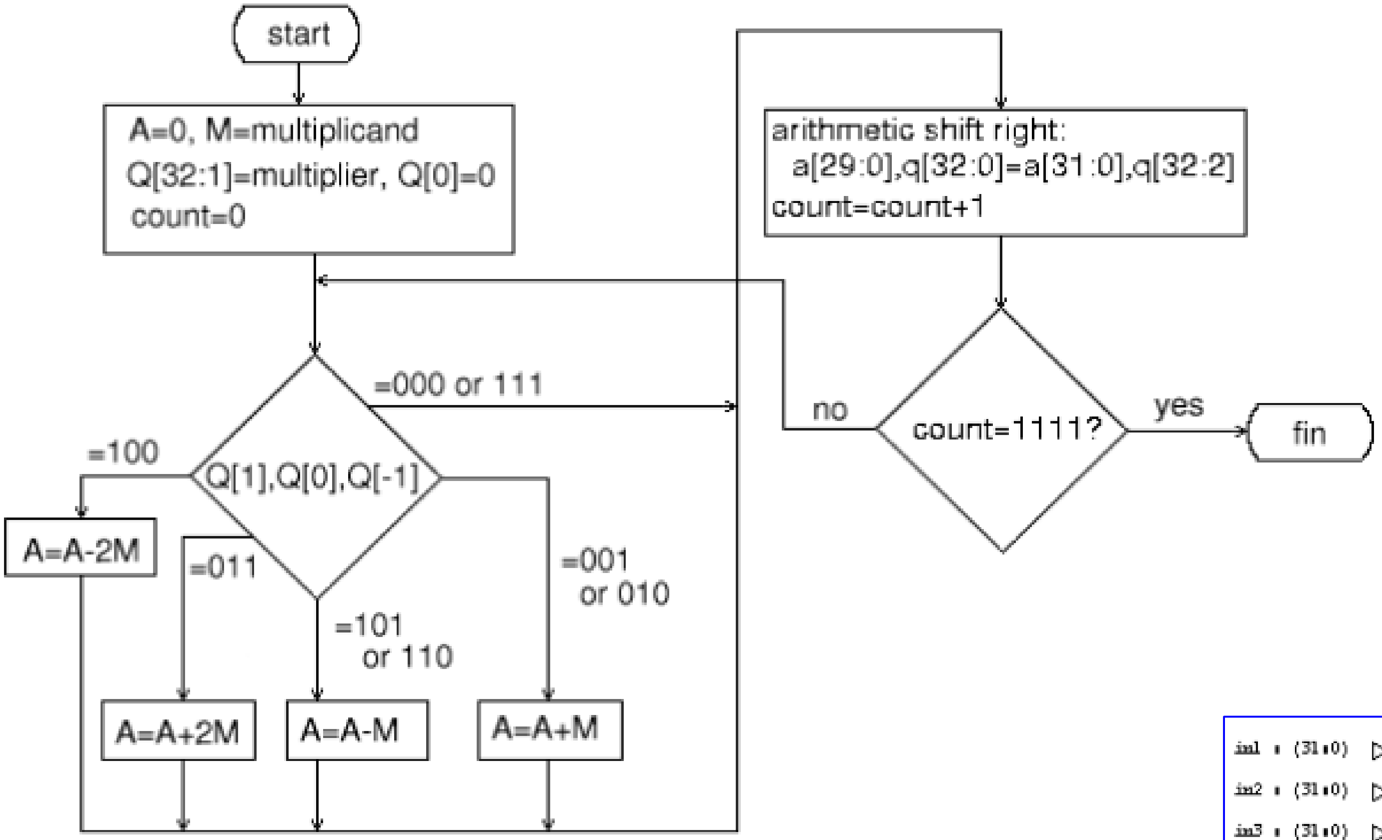
ARM11 MPCore



iPhone processor: 620MHz ARM CPU



Booth Algorithm (Final Course Project)



start

A=0, M=multiplicand
Q[32:1]=multiplier, Q[0]=0
count=0

Q[1],Q[0],Q[-1]

=000 or 111

=100

A=A-2M

=011

A=A+2M

=101
or 110

A=A-M

=001
or 010

A=A+M

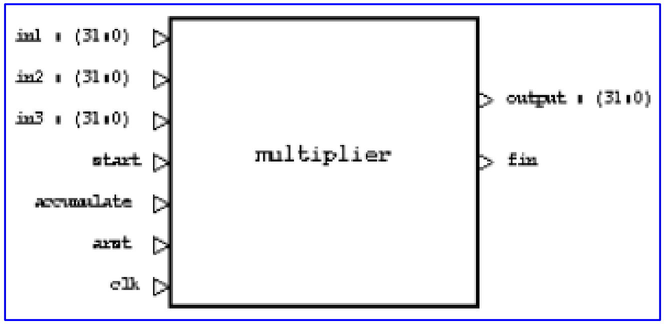
arithmetic shift right:
a[29:0],q[32:0]=a[31:0],q[32:2]
count=count+1

count=1111?

no

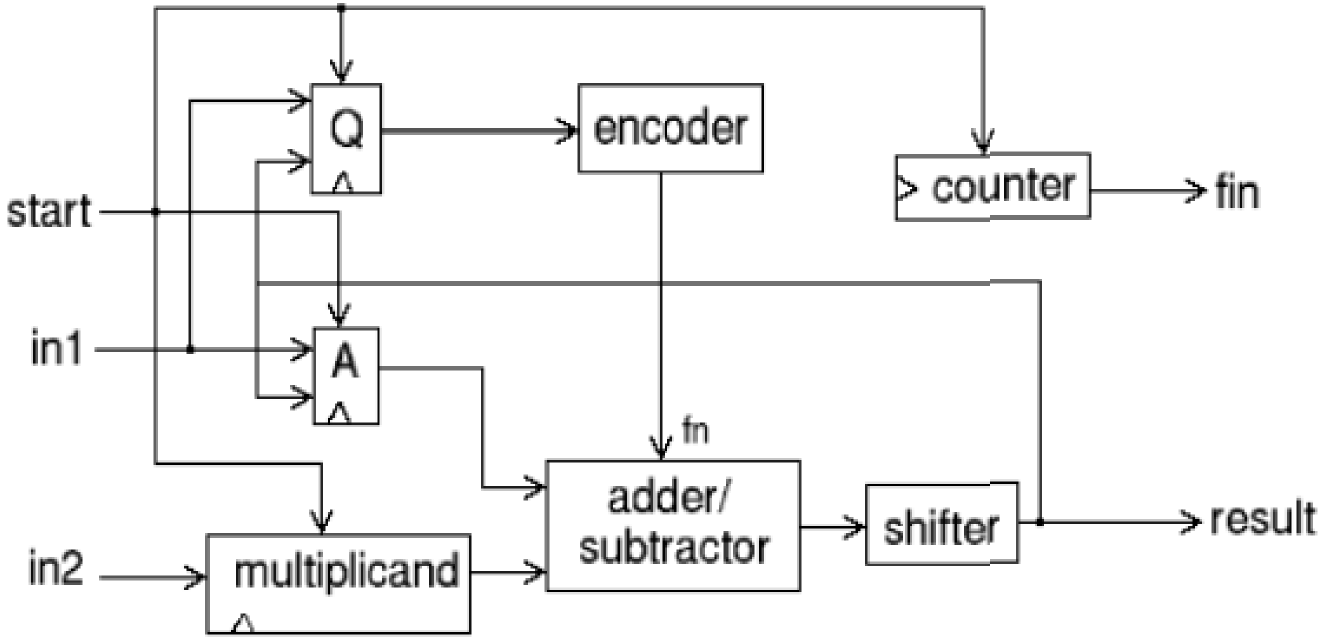
yes

fin



in1 : (31:0)
in2 : (31:0)
in3 : (31:0)
start
accumulate
arst
clk
multiplier
output : (31:0)
fin

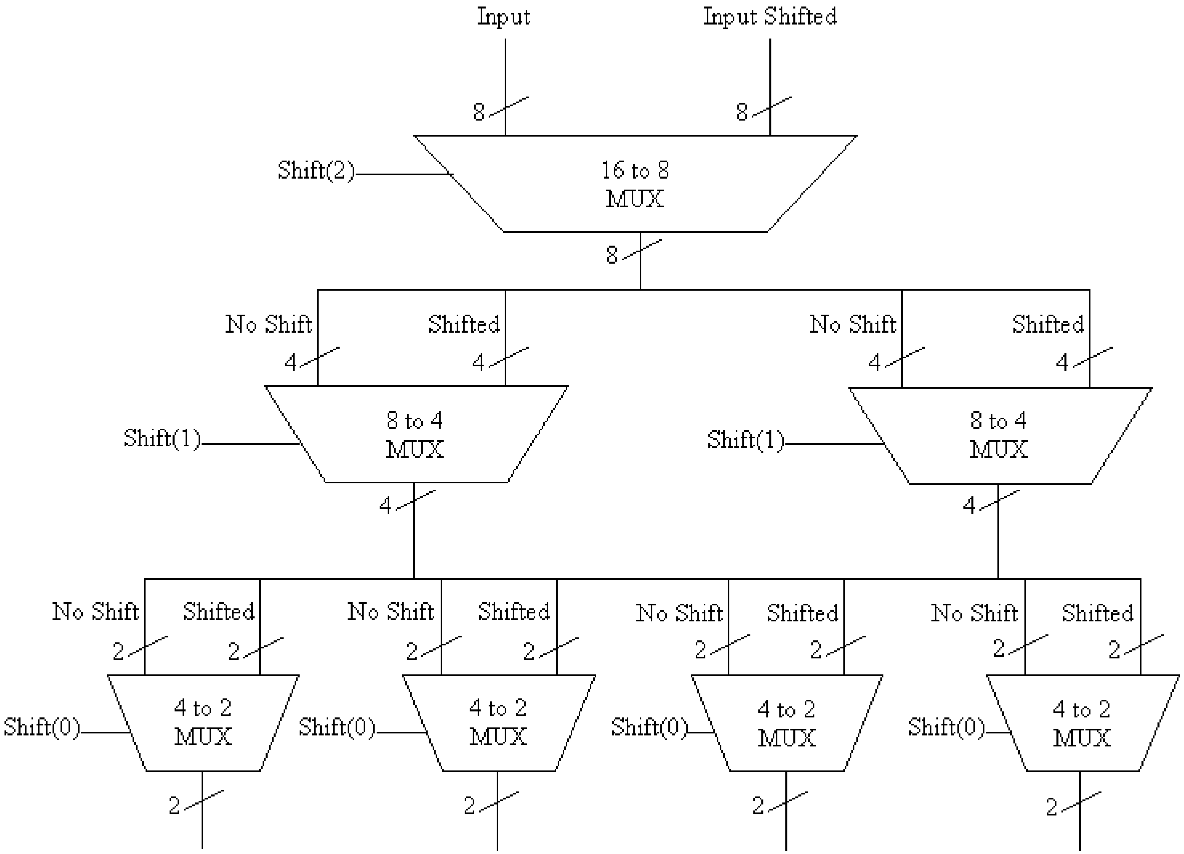
Multiplier Logic

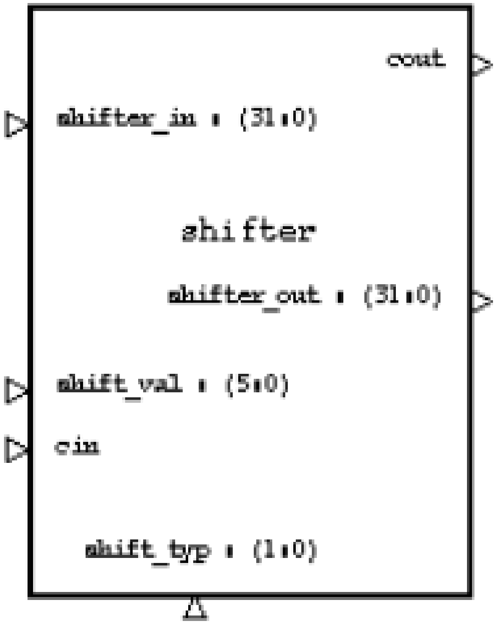


http://aelmahmoudy.users.sourceforge.net/electronix/arm/chapter4.htm

8-bit Barrel Shift (rotate left) diagram

Why is a Barrel Shift- Left ….



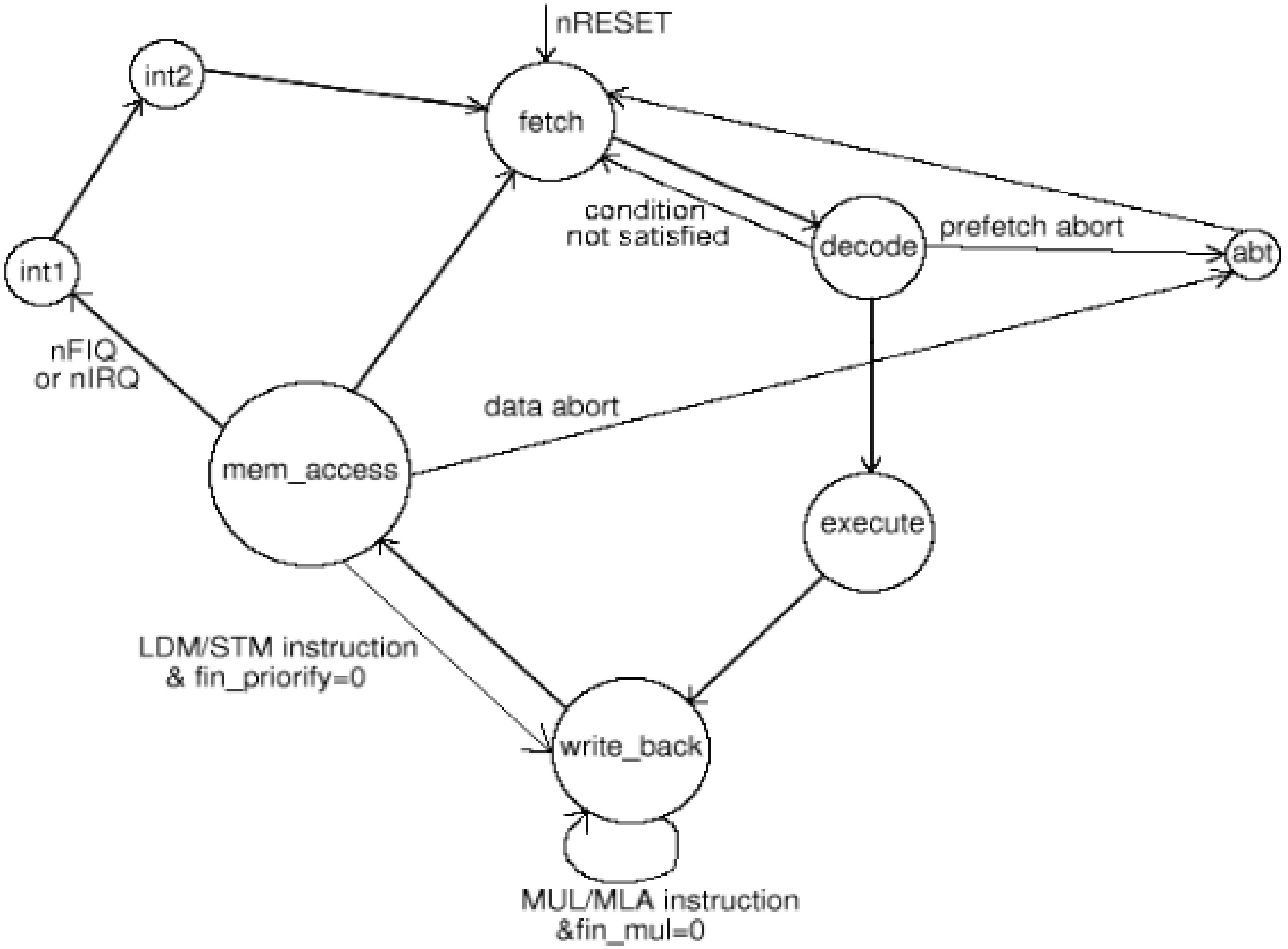


http://www.ece.ualberta.ca/~elliott/ee552/studentAppNotes/2003_w/components/barrel_shifter/

Control Unit

- For any microprocessor, control unit is the heart of the system
- It is responsible for the system operation and so the control unit design is the most important part in the hole design
- Control unit is usually a **pure logic circuit**.

Control Unit: “Finite State Machines”



Computer Organization and Architecture

- A computer programmer can work without needing to know the intimate details of the computer hardware…

- On the other hand …

- Understanding the details of the hardware allows the programmer to optimize the software better for that specific computer…

- Today most of the computer software are not optimal !!! …