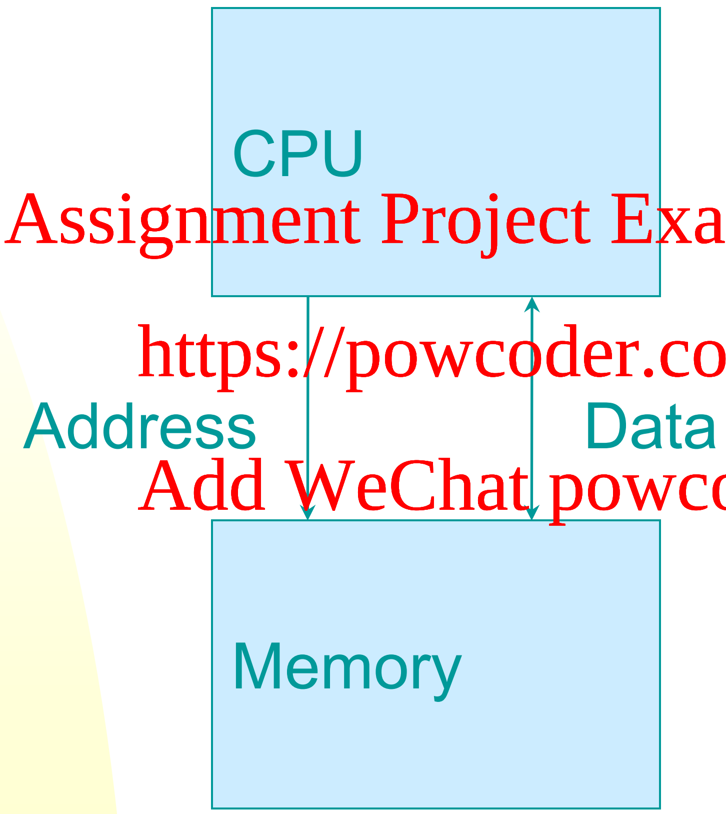
Von Neumann and MIPS

References:

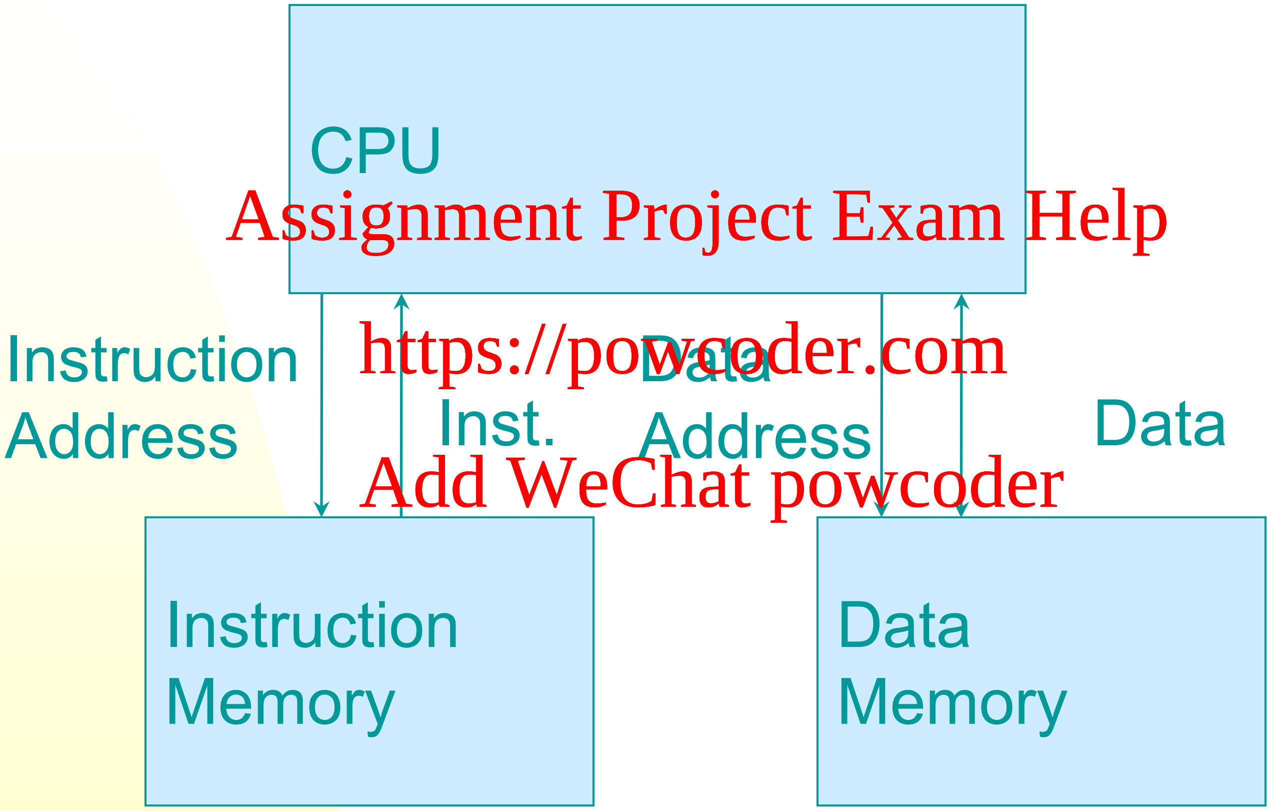
1) MIPS_Vol2.pdf

2) Intro to MIPS Assembly Language Programming

Von Neumann Architecture



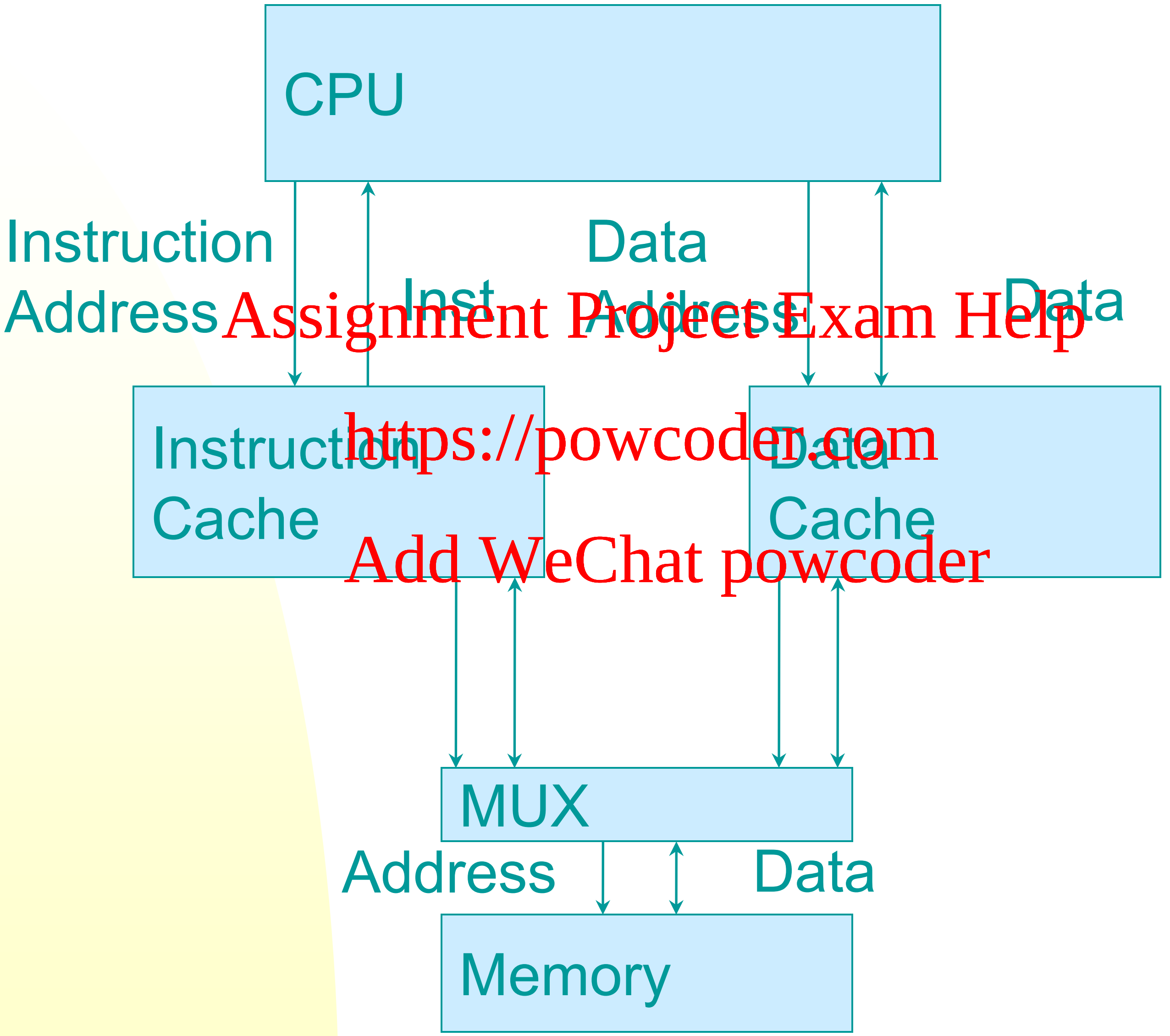
Harvard Architecture



Advantages/Disadvantages

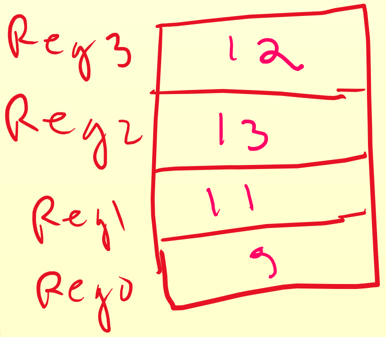
- Advantages
  - Harvard can have different memory sizes
  - Harvard can access both memories at the same time
  - Harvard can have different types of memory
    - Flash for program
    - SRAM for data
  - Instructions can be read-only
- Disadvantages
  - Can run out of one but not the other
  - Requires two memories
  - No self modifying programs?

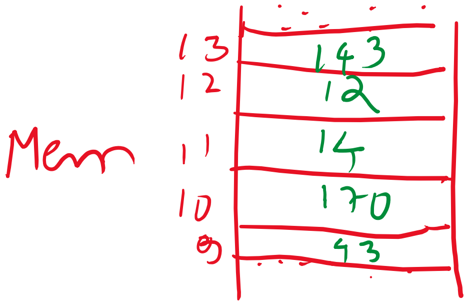
Modified Harvard Architecture



Notations

- Sets of Bits
  - A[3:0] denotes a set of 4 bits: A_3, A_2, A_1, A_0
  - The content of an n-bit register R is referred to as R[n-1:0]
    - R_{n-1} is the most significant bit (MSB), or leftmost bit
    - R_0 is the least significant bit (LSB), or rightmost bit
    - Given R[31:0], R[7:4] refers to the four bits from R_7 down to R_4
- Bit Assignment
  - R2[5:0] ⇐ R1[13:8]
    - Means that bits 5 to 0 of register R2 get assigned the values of bits 13 to 8 of register R1.
- Contents
    - (Reg1) means "content of Reg1"
    - Mem[loc] means "content of memory location loc" (i.e., loc is the address)
    - [Reg1] means the "contents of memory at address in Reg1"



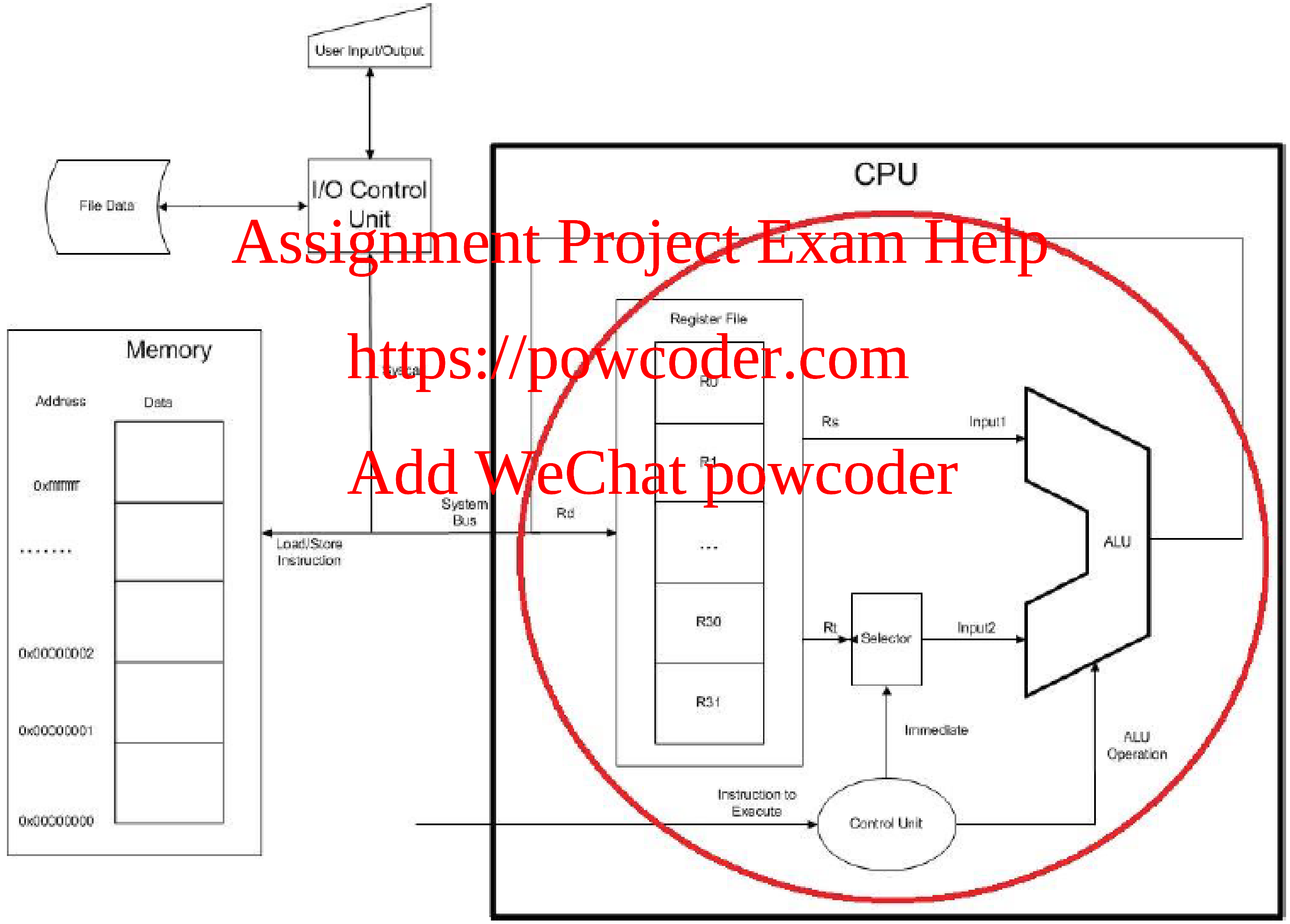


(Reg2) = ?

[Reg2] = ?

Mem[11] = ?

Registers


User Input/Output
File Data
I/O Control Unit
Syscall
CPU
Memory
Address
Data
0xffffffff
....
0x00000002
0x00000001
0x00000000
System Bus
Load/Store Instruction
Rd
Register File
R0
R1
...
R30
R31
Rs
Input1
Rt
Selector
Input2
ALU
Immediate
ALU Operation
Instruction to Execute
Control Unit

MIPS Registers

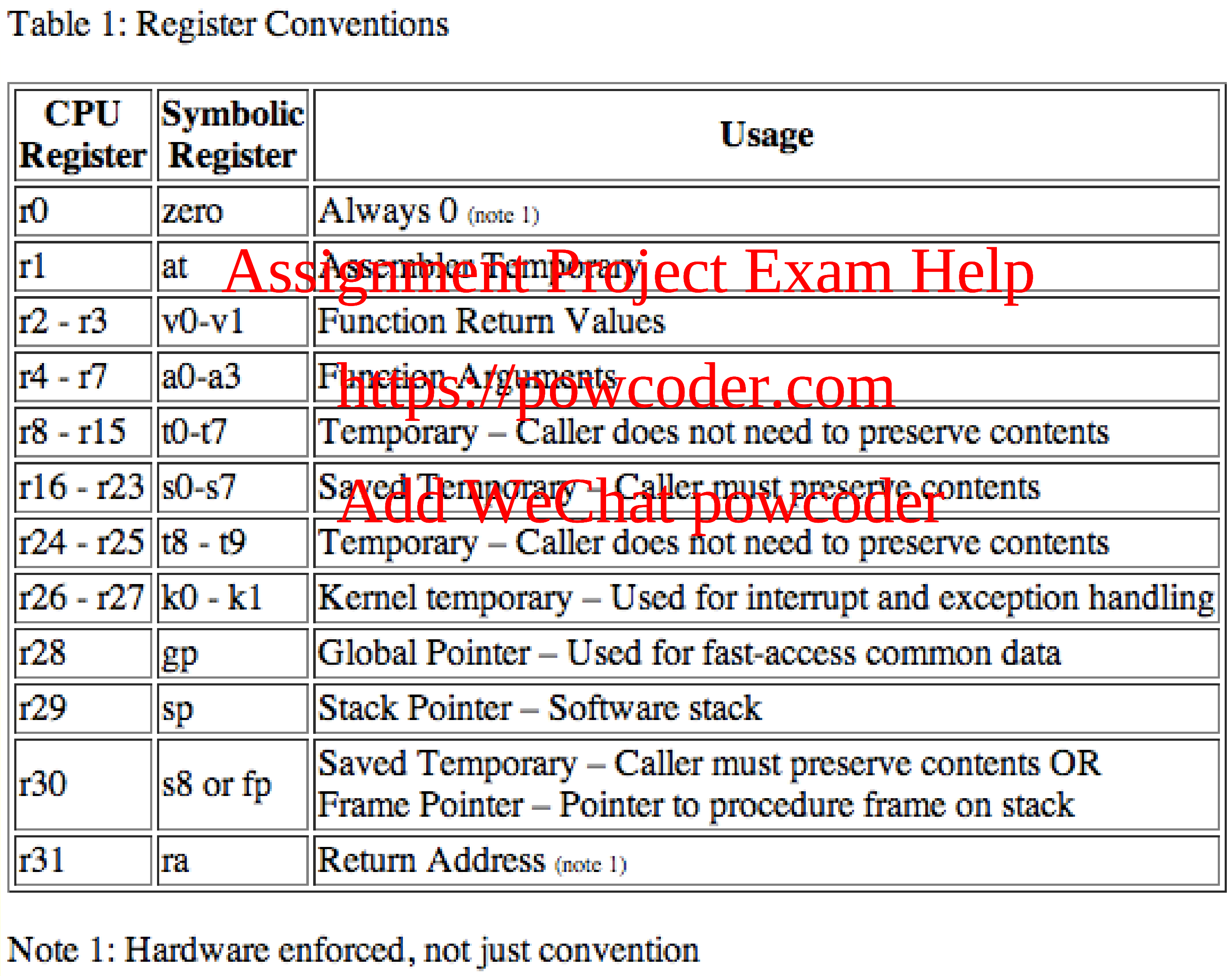


Table 1: Register Conventions

CPU Register	Symbolic Register	Usage
r0	zero	Always 0 (note 1)
r1	at	Assembler Temporary
r2 - r3	v0-v1	Function Return Values
r4 - r7	a0-a3	Function Arguments
r8 - r15	t0-t7	Temporary – Caller does not need to preserve contents
r16 - r23	s0-s7	Saved Temporary – Caller must preserve contents
r24 - r25	t8 - t9	Temporary – Caller does not need to preserve contents
r26 - r27	k0 - k1	Kernel temporary – Used for interrupt and exception handling
r28	gp	Global Pointer – Used for fast-access common data
r29	sp	Stack Pointer – Software stack
r30	s8 or fp	Saved Temporary – Caller must preserve contents OR Frame Pointer – Pointer to procedure frame on stack
r31	ra	Return Address (note 1)

Note 1: Hardware enforced, not just convention

Register File (2)

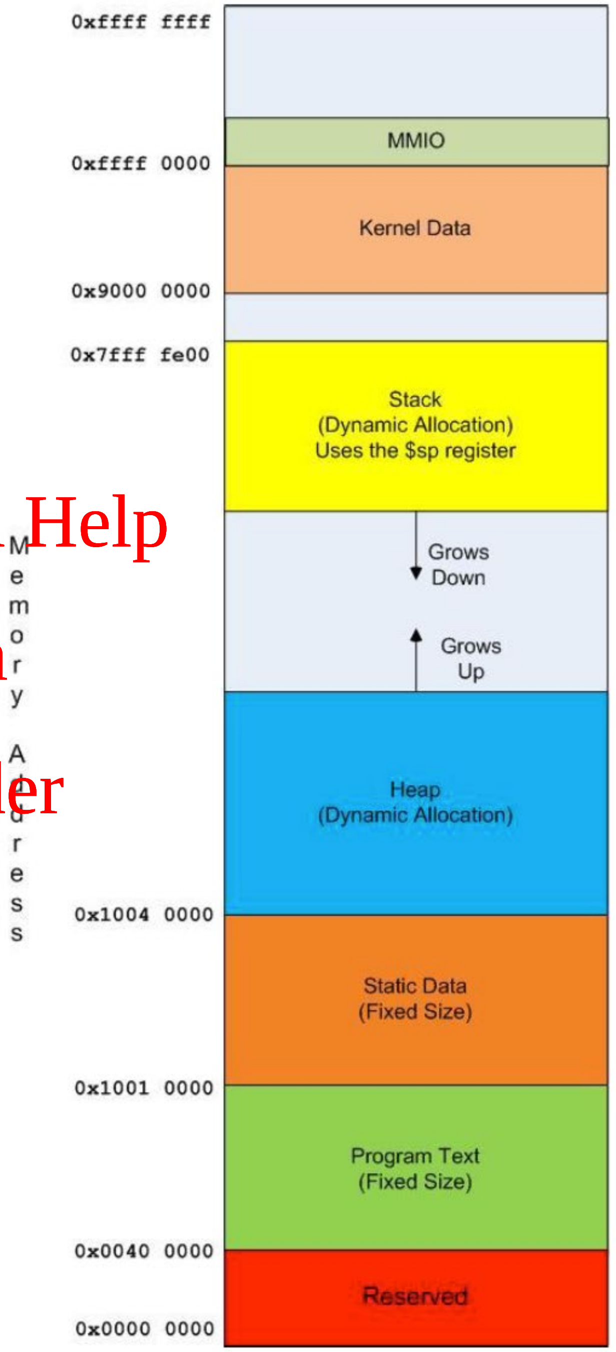
- Temporary can be used $t0…$t9
  - Procedures can modify these
- Saved can be used $s0..$s7, but are caller save
  - Procedures must save/restore these
- $zero (or $0) is always 0
- $a0..a3 Are passed to functions as parameters
- $v0..v1 Are returned from functions

MIPS Memory

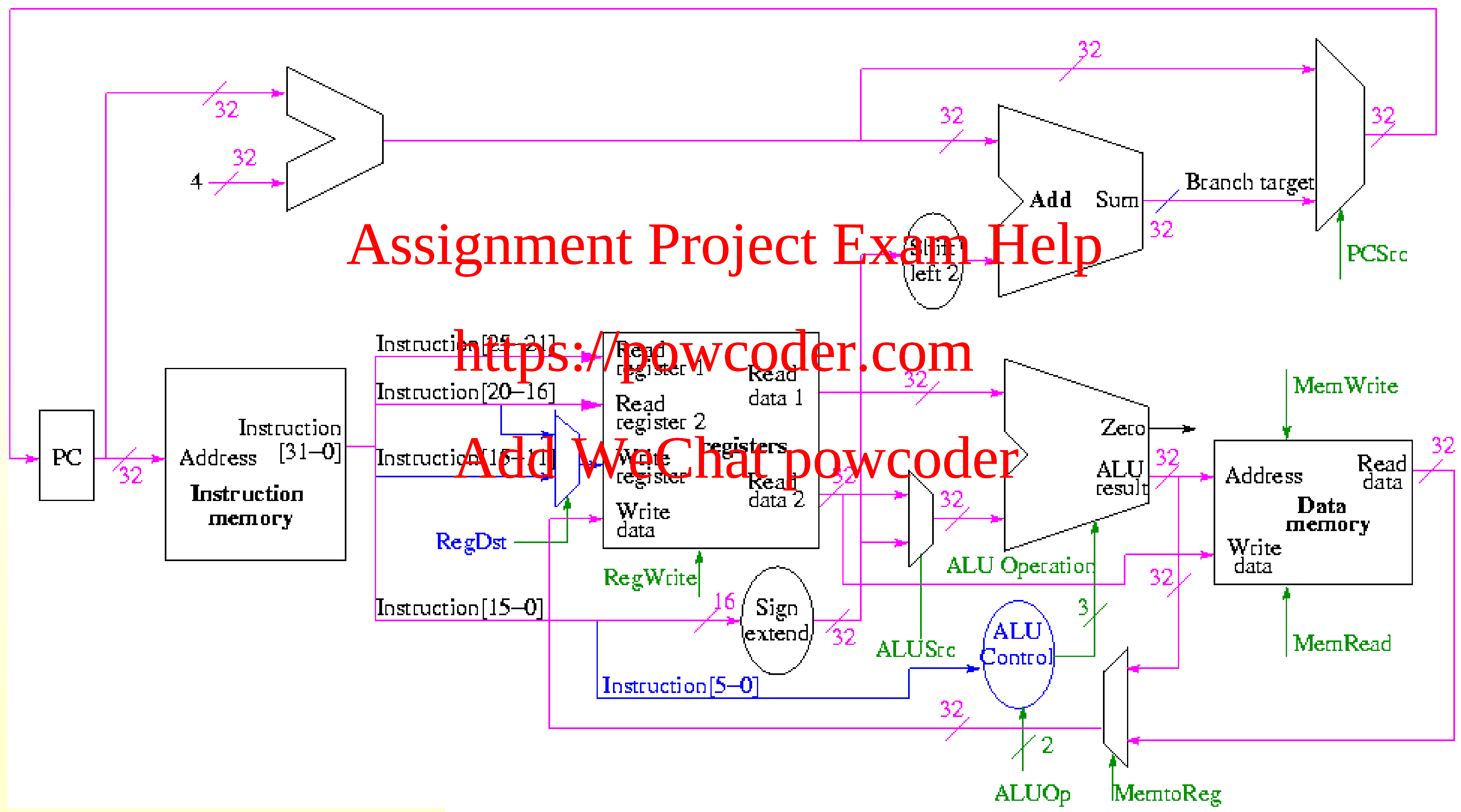
- 32-bit "flat" memory model
  - Address 0x00000000 to 0xFFFFFFFF
  - How much memory is that?
  - What if I want less?
  - What if I want more?

Memory “Segments”

- Three segments for now
  - Reserved:
    - 0x0000_0000 to 0x0040_0000
    - Special code for I/O and OS
  - Program Text:
    - 0x0040_0000 to 0x1000_0000
    - Machine code for your instructions
  - Static Data
    - 0x1001_0000 to 0x1004_0000
    - Data that is allocated before your program runs
- What is a segmentation fault?



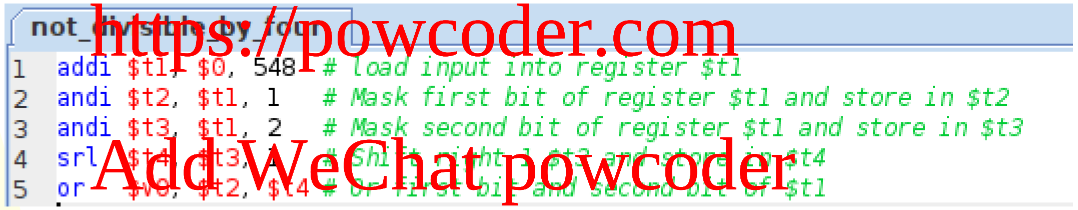
MIPS Data Path (much more later!)



MIPS Example

Not Divisible By Four Program

- Sequence of 5 Instructions

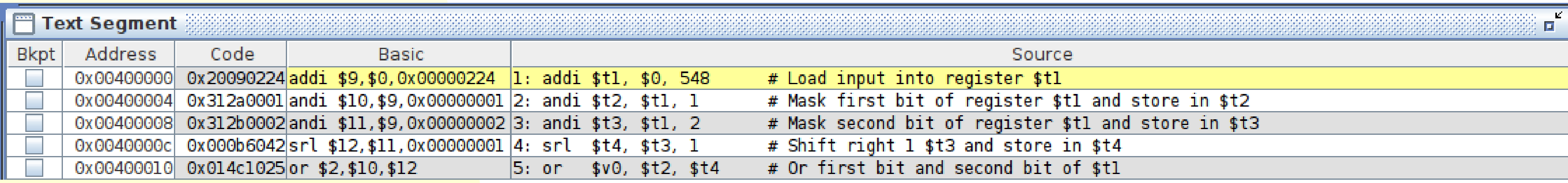


not_divisible_by_four

```
addi $t1, $0, 548  # load input into register $t1
andi $t2, $t1, 1   # Mask first bit of register $t1 and store in $t2
andi $t3, $t1, 2   # Mask second bit of register $t1 and store in $t3
srl  $t4, $t3, 1   # Shift right 1 $t3 and store in $t4
or   $v0, $t2, $t4 # Or first bit and second bit of $t1
```

Memory Unit

- Instructions stored in memory.
- The first instruction is always at 0x0040 0000
- Each instruction is 32 bits long (4 Bytes, 1 byte = 8 bits)
- 5 instructions = 5 * 4 bytes = 20 bytes (size of the program).
- Instructions stored sequentially in memory with the address of the next instruction being +4 of the previous instruction



Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x20090224	addi $9,$0,0x00000224	1: addi $t1, $0, 548 # Load input into register $t1
☐	0x00400004	0x312a0001	andi $10,$9,0x00000001	2: andi $t2, $t1, 1 # Mask first bit of register $t1 and store in $t2
☐	0x00400008	0x312b0002	andi $11,$9,0x00000002	3: andi $t3, $t1, 2 # Mask second bit of register $t1 and store in $t3
☐	0x0040000c	0x000b6042	srl $12,$11,0x00000001	4: srl $t4, $t3, 1 # Shift right 1 $t3 and store in $t4
☐	0x00400010	0x014c1025	or $2,$10,$12	5: or $v0, $t2, $t4 # Or first bit and second bit of $t1

Instruction in Binary

- addi $t1, $0, 548 - 0x20090224

  0010 0000 0000 1001 0000 0010 0010 0100

- addi is an I-Type instruction (immediate type)
  - op rt, rs, imm:
    - rt - destination, rs - source 1, imm - immediate

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

Instruction in Binary

- addi $t1, $0, 548 - 0x20090224

  0010 0000 0000 1001 0000 0010 0010 0100

- addi is an I-Type instruction (immediate type)
  - op rt, rs, imm:
    - rt - destination, rs - source 1, imm - immediate

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

001000	rs - 5 bits	rt - 5 bits	imm - 16 bits

Instruction in Binary

- addi $t1, $0, 548 - 0x20090224

  0010 0000 0000 1001 0000 0010 0010 0100

- addi is an I-Type instruction (immediate-type)
  - op rt, rs, imm:
    - rt - destination, rs - source 1, imm - immediate

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

001000	00000	rt - 5 bits	imm - 16 bits

Instruction in Binary

- addi $t1, $0, 548 - 0x20090224

  0010 0000 0000 1001 0000 0010 0010 0100

- addi is an I-Type instruction (immediate type)
  - op rt, rs, imm:
    - rt - destination, rs - source 1, imm - immediate

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

001000	00000	01001	imm - 16 bits

Instruction in Binary

- addi $t1, $0, 548 - 0x20090224

  0010 0000 0000 1001 0000 0010 0010 0100

- addi is an I-Type instruction (immediate type)
  - op rt, rs, imm:
    - rt - destination, rs - source 1, imm - immediate

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

001000	00000	01001	0000 0010 0010 0100

rs and rt are Register Addresses

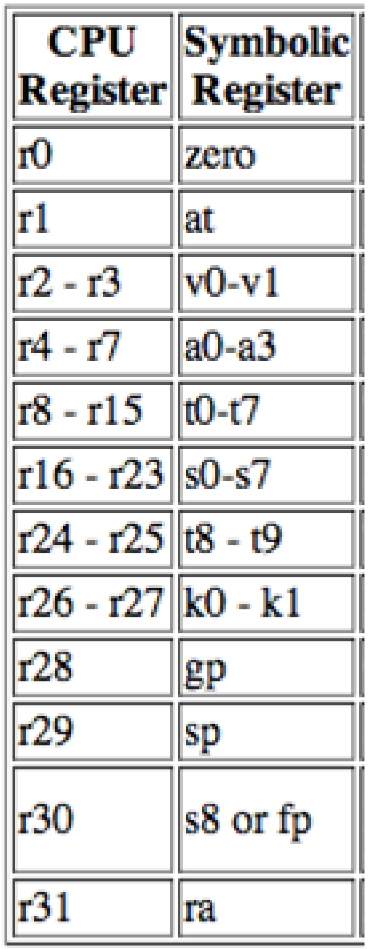
- addi $t1, $0, 548 - 0x20090224

  0010 0000 0000 1001 0000 0010 0010 0100

- addi is an I-Type instruction
  - op rt, rs, imm:
    - source, destination,

- rs = $0 - register 0
- rt = $t1 - register 9

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits
001000	00000	01001	0000 0010 0010 0100



CPU Register	Symbolic Register
r0	zero
r1	at
r2 - r3	v0-v1
r4 - r7	a0-a3
r8 - r15	t0-t7
r16 - r23	s0-s7
r24 - r25	t8 - t9
r26 - r27	k0 - k1
r28	gp
r29	sp
r30	s8 or fp
r31	ra

Introduction to MIPS Assembly

References:

1) MIPS_Vol2.pdf

2) Intro to MIPS Assembly Language Programming

MIPS Overview

- Different Type of Instructions (so far we know R-type and I-Type, will learn J-type soon)
- Natively 2’s complement for representing binary numbers (unsigned options available)
- Different addressing modes (will learn soon):
  - Immediate (non-memory addressing mode)
  - Register (non-memory addressing mode)
  - Direct, Indirect & Base+Offset (memory addressing modes)

MIPS Instruction Review

I-type instructions:

- addi $t1, $0, 548
- op rs, rt, imm:
  - rt - destination, rs - source 1, imm - immediate

op - 6 bits	rs - 5 bits	rt - 5 bits	imm - 16 bits

R-type instructions:

- or $v0, $t2, $t4
- op rs, rt, rd, shift amount, funct:
  - ★ rd - destination, rs - source 1, rt source 2

op - 6 bits	rs - 5 bits	rt - 5 bits	rd - 5 bits	shamt - 5 bits	funct - 6 bits

MIPS Overview - Commands

MIPS instructions can be broken down into 3 categories:

- Data Movement
  - Move data between memory and registers
    - For example: lw is load word, sw is store word
- Operate
  - Manipulate data directly
    - For example: add is addition, xor is logical
- Control
  - Change the sequence of instruction execution
    - For example: b is branch, jal is jump and link, ret is return

MIPS Overview - Commands

MIPS instructions can be broken down into 3 categories:

- Data Movement
  - Move data between memory and registers
    - For example: lw is load word, sw is store word
- Operate
  - Manipulate data directly
    - For example: add is addition, xor is logical
- Control
  - Change the sequence of instruction execution
    - For example: b is branch, jal is jump and link, ret is return

Operate: R-Type Logical Instructions

- AND, OR, NOR, XOR
  - Uses bit-wise logical operation
  - Example:
    - AND $t1, $t2, $t3
    - $t1 = $t2 & $t3
      - For all 32 bits…
        - Bit 0 of $t2 ANDed with bit 0 of $t3 and put in bit 0 of $t1

Operate: Arithmetic Instructions

- Arithmetic Signed
  - Uses 2's complement integers
  - Example:
    - ADD $t1, $t2, $t3
    - $t1 = $t2 + $t3

Operate: Shift Logical

- Shift left or right logical
  - Number of bits to shift is shamt value
- Example:
  - SLL $t1, $t2, 4
    - $t1 = $t2 << 4
  - SRL $t1, $t2, 2
    - $t1 = $t2 >> 2
- Lower bits dropped in SRL
  - Equivalent to integer divide by 2 for each bit shifted
- Zeros inserted in SLL or SRL
  - Equivalent to integer multiply by 2 for each bit shifted

Operate: Shift Logical Variable

- Shift left or right logical
  - Similar behavior to Shift Logical
  - Except number of bits to shift is in **register**
- Example:
  - SLLV $t1, $t2, $t3
    - $t1 = $t2 << $t3
  - SRLV $t1, $t2, $t3
    - $t1 = $t2 >> $t3

Operate: Shift Arithmetic (and Variable)

- Similar to Logical and Logical Variable Shifts
  - Except right shifts extend the sign bit
- Examples:
  - SRA $t1, $t2, 3
    - $t1 = sign_extend($t2 >> 3)
  - SRAV $t1, $t2, $t3
    - $t1 = sign_extend($t2 >> $t3)

Set Operations

- Set on less than
  - Two registers (SLT and SLTU)
  - Register and Immediate (SLTI and SLTIU)
  - Unsigned or Signed (SLT/SLTI vs SLTU/SLTIU)
- Destination register is 0 or 1 decimal
- Example:
  - SLT $t1, $t2, $t3
  - Sets $t1 to 1 if $t2<$t3
  - Otherwise, $t1 is 0