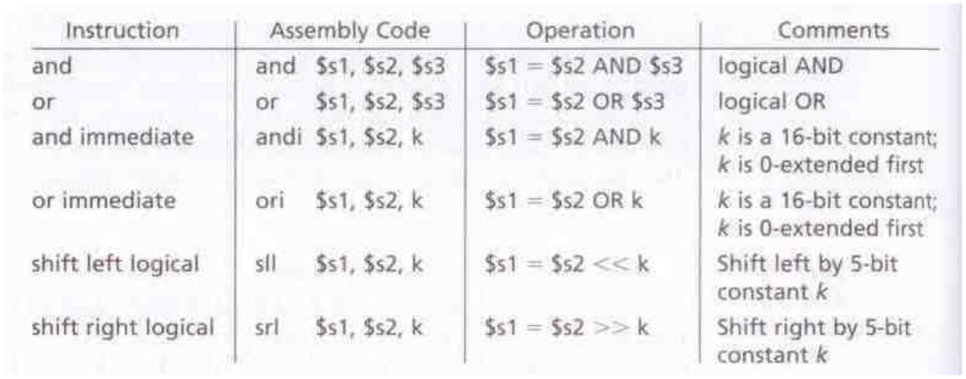
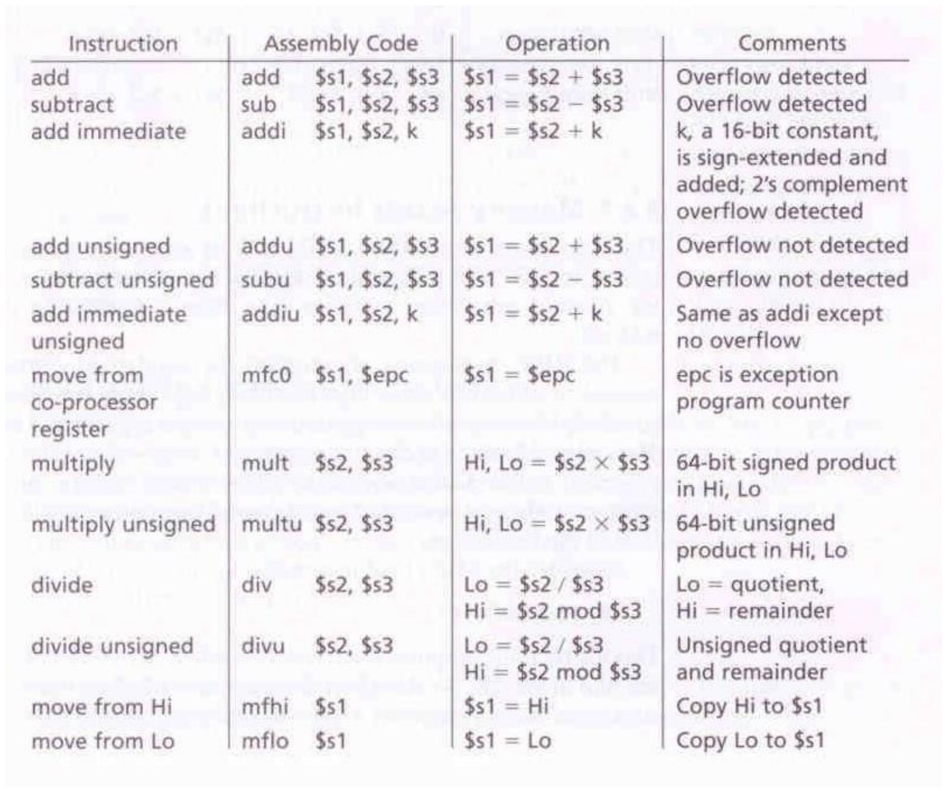
Design Example: Design of a RISI Processor (MIPS R2000)

- CISC: Complex Instruction Set Computing, Intel 80x86, Motorola 68000/68020
    - A variety of powerful instructions and addressing modes
- RISC: Reduced Instruction Set Computing, IBM 801, RS/6000, MIPS R2000 (Stanford), Sun Micro SPARC.
    - Use a small and simple set of instructions rather than a variety of complex instructions and versatile addressing modes
- MIPS
    - Performance metric: Millions of Instructions Per Second
    - Microprocessor without hardware Interlock Processing System
- Characterises of most RISC Processors
    - Uniform instruction length
    - Few instruction formats
    - Few addressing modes
    - Large number of registers (RISC also called register-register architectures)
        - CISC typically 8-12 vs. RISC often 32 registers
    - Load and store architecture
        - Only load and store instructions can access memory
        - Key idea: the absence of arithmetic instructions that directly operate on memory operands
    - No implied operands (No side-effects)
- MIPS Instruction Set Architecture (ISA)
    - Three address format for ALU instructions
        - E.g., add $5, $3, $4
        - Specify two sources addresses and one destination address
        - Total 32 general purpose registers ($0, ... $31)
    - Logical Instructions



Instruction	Assembly Code	Operation	Comments
and	and $s1, $s2, $s3	$s1 = $s2 AND $s3	logical AND
or	or $s1, $s2, $s3	$s1 = $s2 OR $s3	logical OR
and immediate	andi $s1, $s2, k	$s1 = $s2 AND k	*k* is a 16-bit constant; *k* is 0-extended first
or immediate	ori $s1, $s2, k	$s1 = $s2 OR k	*k* is a 16-bit constant; *k* is 0-extended first
shift left logical	sll $s1, $s2, k	$s1 = $s2 << k	Shift left by 5-bit constant *k*
shift right logical	srl $s1, $s2, k	$s1 = $s2 >> k	Shift right by 5-bit constant *k*

- Arithmetic Instructions



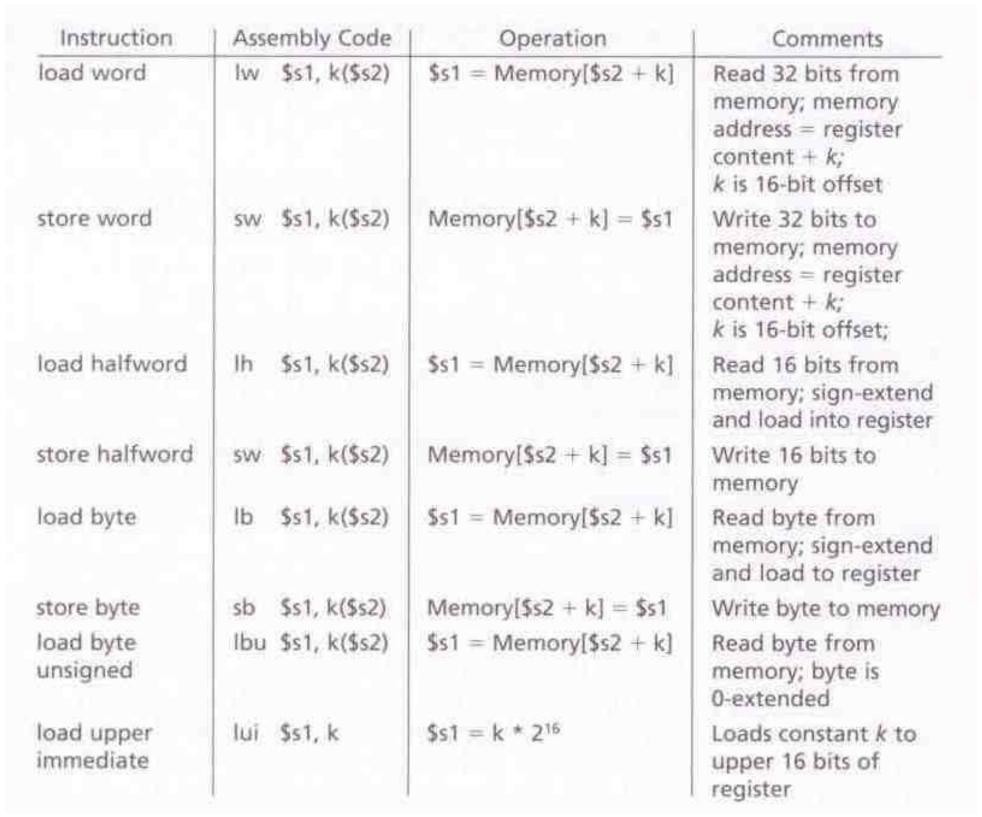
Instruction	Assembly Code	Operation	Comments
add	add $s1, $s2, $s3	$s1 = $s2 + $s3	Overflow detected
subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3	Overflow detected
add immediate	addi $s1, $s2, k	$s1 = $s2 + k	k, a 16-bit constant, is sign-extended and added; 2's complement overflow detected
add unsigned	addu $s1, $s2, $s3	$s1 = $s2 + $s3	Overflow not detected
subtract unsigned	subu $s1, $s2, $s3	$s1 = $s2 − $s3	Overflow not detected
add immediate unsigned	addiu $s1, $s2, k	$s1 = $s2 + k	Same as addi except no overflow
move from co-processor register	mfc0 $s1, $epc	$s1 = $epc	epc is exception program counter
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

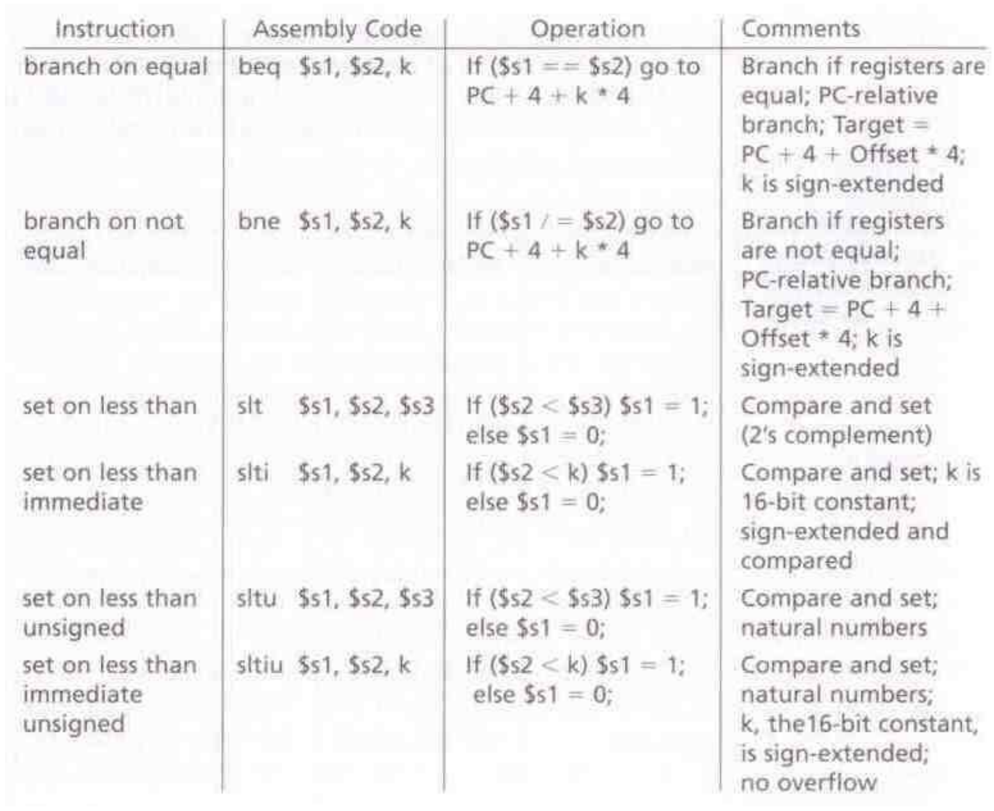
move from Hi	mfhi $s1	$s1 = Hi	Copy Hi to $s1
move from Lo	mflo $s1	$s1 = Lo	Copy Lo to $s1

- Memory Access Instructions



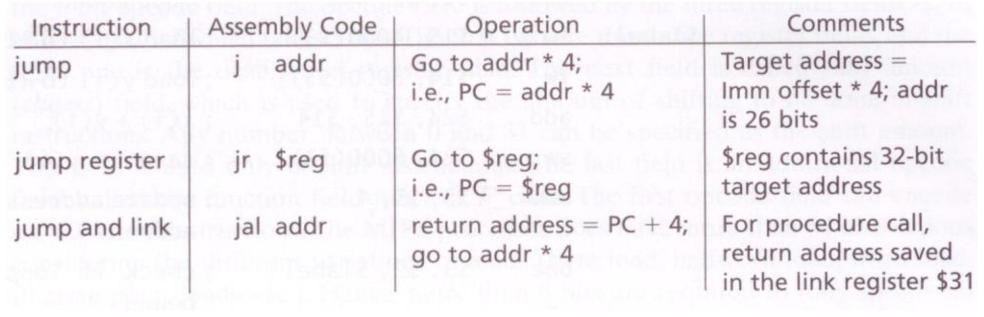
Instruction	Assembly Code	Operation	Comments
load word	lw $s1, k($s2)	$s1 = Memory[$s2 + k]	Read 32 bits from memory; memory address = register content + *k*; *k* is 16-bit offset
store word	sw $s1, k($s2)	Memory[$s2 + k] = $s1	Write 32 bits to memory; memory address = register content + *k*; *k* is 16-bit offset;
load halfword	lh $s1, k($s2)	$s1 = Memory[$s2 + k]	Read 16 bits from memory; sign-extend and load into register
store halfword	sw $s1, k($s2)	Memory[$s2 + k] = $s1	Write 16 bits to memory
load byte	lb $s1, k($s2)	$s1 = Memory[$s2 + k]	Read byte from memory; sign-extend and load to register
store byte	sb $s1, k($s2)	Memory[$s2 + k] = $s1	Write byte to memory
load byte unsigned	lbu $s1, k($s2)	$s1 = Memory[$s2 + k]	Read byte from memory; byte is 0-extended
load upper immediate	lui $s1, k	$s1 = k * 2^{16}	Loads constant *k* to upper 16 bits of register

- Control Transfer Instructions



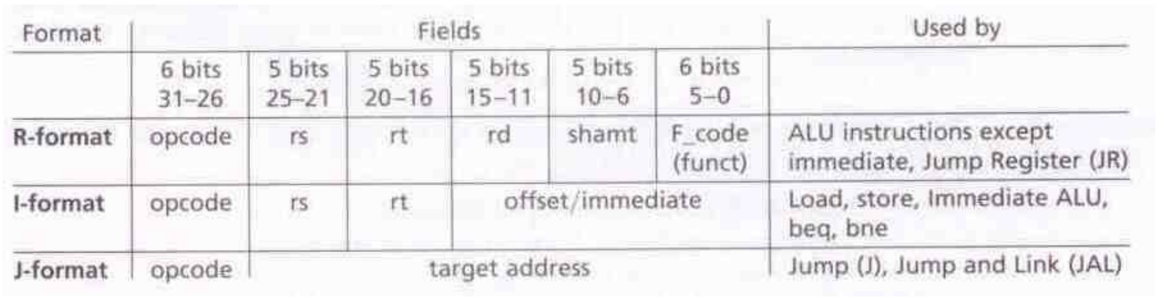
Instruction	Assembly Code	Operation	Comments
branch on equal	beq $s1, $s2, k	If ($s1 == $s2) go to PC + 4 + k * 4	Branch if registers are equal; PC-relative branch; Target = PC + 4 + Offset * 4; k is sign-extended
branch on not equal	bne $s1, $s2, k	If ($s1 / = $s2) go to PC + 4 + k * 4	Branch if registers are not equal; PC-relative branch; Target = PC + 4 + Offset * 4; k is sign-extended
set on less than	slt $s1, $s2, $s3	If ($s2 < $s3) $s1 = 1; else $s1 = 0;	Compare and set (2's complement)
set on less than immediate	slti $s1, $s2, k	If ($s2 < k) $s1 = 1; else $s1 = 0;	Compare and set; k is 16-bit constant; sign-extended and compared
set on less than unsigned	sltu $s1, $s2, $s3	If ($s2 < $s3) $s1 = 1; else $s1 = 0;	Compare and set; natural numbers
set on less than immediate unsigned	sltiu $s1, $s2, k	If ($s2 < k) $s1 = 1; else $s1 = 0;	Compare and set; natural numbers; k, the16-bit constant, is sign-extended; no overflow

- Unconditional Control Transfer Instructions



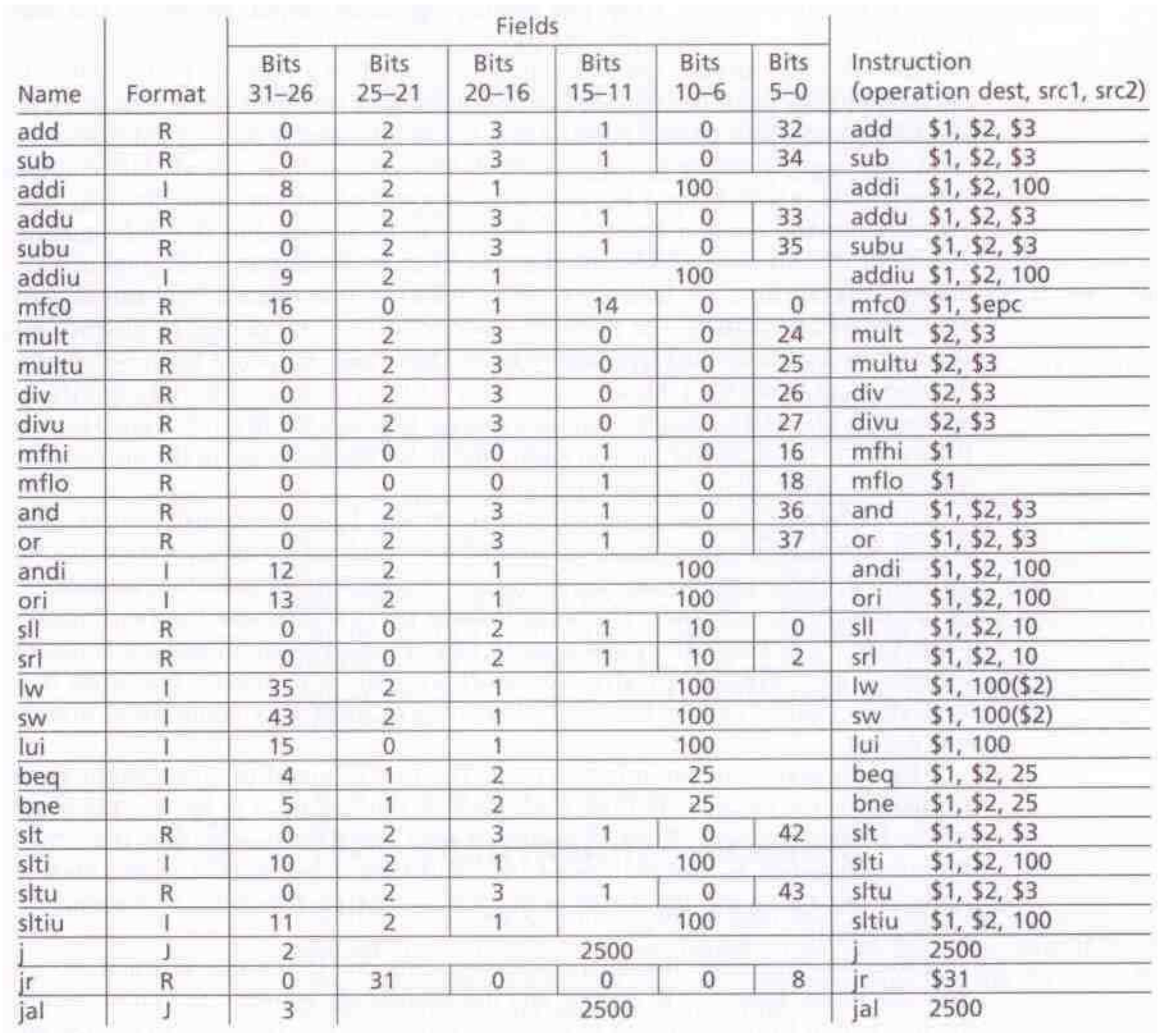
Instruction	Assembly Code	Operation	Comments
jump	j addr	Go to addr * 4; i.e., PC = addr * 4	Target address = Imm offset * 4; addr is 26 bits
jump register	jr $reg	Go to $reg; i.e., PC = $reg	$reg contains 32-bit target address
jump and link	jal addr	return address = PC + 4; go to addr * 4	For procedure call, return address saved in the link register $31

- MIPS Instruction Encoding
  - Three different instruction formats: R-format, I-format, J-format
    - R-Format: ALU instructions require three operands. And the jump register instruction.
    - I-Format: arithmetic instructions, load and store instructions, and branch instructions that need an immediate constant in the instruction.
    - J-Format: jump instructions.



Format	Fields						Used by
	6 bits 31–26	5 bits 25–21	5 bits 20–16	5 bits 15–11	5 bits 10–6	6 bits 5–0	
R-format	opcode	rs	rt	rd	shamt	F_code (funct)	ALU instructions except immediate, Jump Register (JR)
I-format	opcode	rs	rt	offset/immediate			Load, store, Immediate ALU, beq, bne
J-format	opcode	target address					Jump (J), Jump and Link (JAL)

- o Instruction Encoding and Decoding

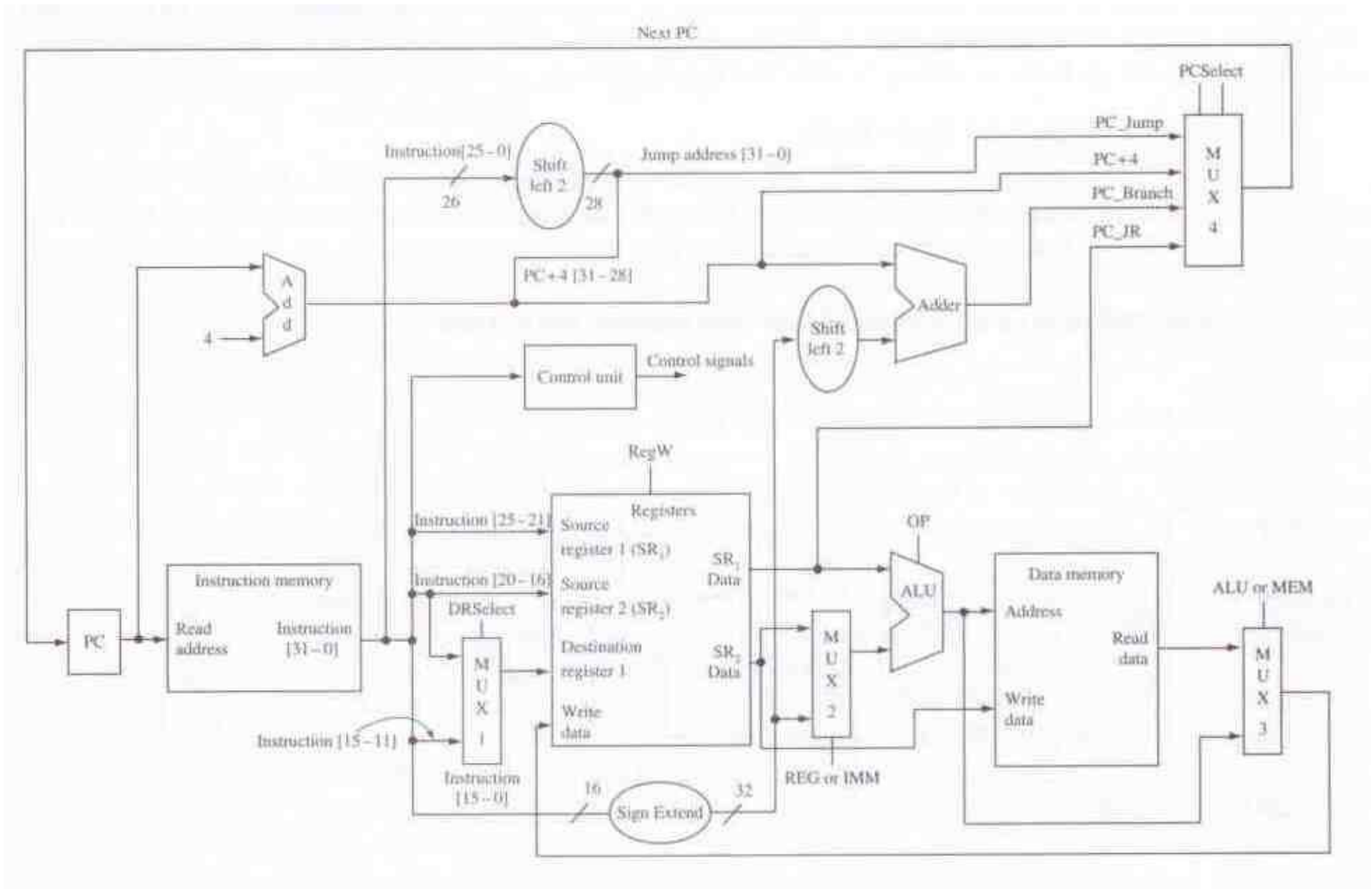


		Fields						
Name	Format	Bits 31–26	Bits 25–21	Bits 20–16	Bits 15–11	Bits 10–6	Bits 5–0	Instruction (operation dest, src1, src2)
add	R	0	2	3	1	0	32	add $1, $2, $3
sub	R	0	2	3	1	0	34	sub $1, $2, $3
addi	I	8	2	1		100		addi $1, $2, 100
addu	R	0	2	3	1	0	33	addu $1, $2, $3
subu	R	0	2	3	1	0	35	subu $1, $2, $3
addiu	I	9	2	1		100		addiu $1, $2, 100
mfc0	R	16	0	1	14	0	0	mfc0 $1, $epc
mult	R	0	2	3	0	0	24	mult $2, $3
multu	R	0	2	3	0	0	25	multu $2, $3
div	R	0	2	3	0	0	26	div $2, $3
divu	R	0	2	3	0	0	27	divu $2, $3
mfhi	R	0	0	0	1	0	16	mfhi $1
mflo	R	0	0	0	1	0	18	mflo $1
and	R	0	2	3	1	0	36	and $1, $2, $3
or	R	0	2	3	1	0	37	or $1, $2, $3
andi	I	12	2	1		100		andi $1, $2, 100
ori	I	13	2	1		100		ori $1, $2, 100
sll	R	0	0	2	1	10	0	sll $1, $2, 10
srl	R	0	0	2	1	10	2	srl $1, $2, 10
lw	I	35	2	1		100		lw $1, 100($2)
sw	I	43	2	1		100		sw $1, 100($2)
lui	I	15	0	1		100		lui $1, 100
beq	I	4	1	2		25		beq $1, $2, 25
bne	I	5	1	2		25		bne $1, $2, 25
slt	R	0	2	3	1	0	42	slt $1, $2, $3
slti	I	10	2	1		100		slti $1, $2, 100
sltu	R	0	2	3	1	0	43	sltu $1, $2, $3
sltiu	I	11	2	1		100		sltiu $1, $2, 100
j	J	2			2500			j 2500
jr	R	0	31	0	0	0	8	jr $31
jal	J	3			2500			jal 2500

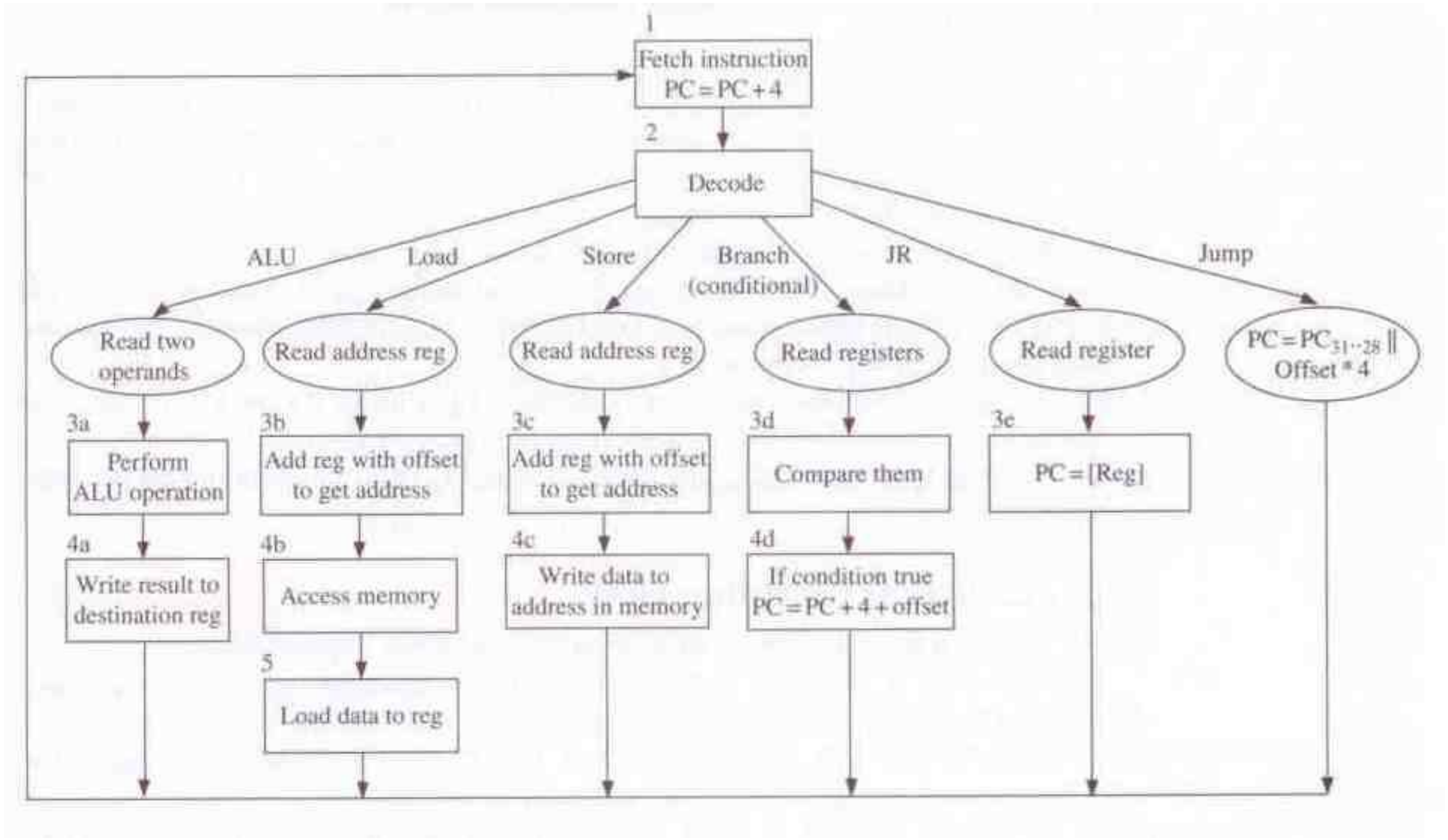
- o Example:
  - ▪ andi $3, $3, 0 (Hex: 30 63 00 00)
  - ▪ lw $15, 4000($3) (Hex: 8C 6F 0F A0)
  - ▪ bne $3, $2, -6 (Hex: 14 62 FF FA)
- Implementation of a MIPS Subset
  - o Arithmetic
    - ▪ Add, subtract, add immediate

- Logic
  - And, or, and immediate, or immediate, shift left logical, shift right logical
- Data transfer
  - Load word, store word
- Conditional branch
  - Branch on equal, branch on not equal, set on less than
- Unconditional branch
  - Jump, jump register

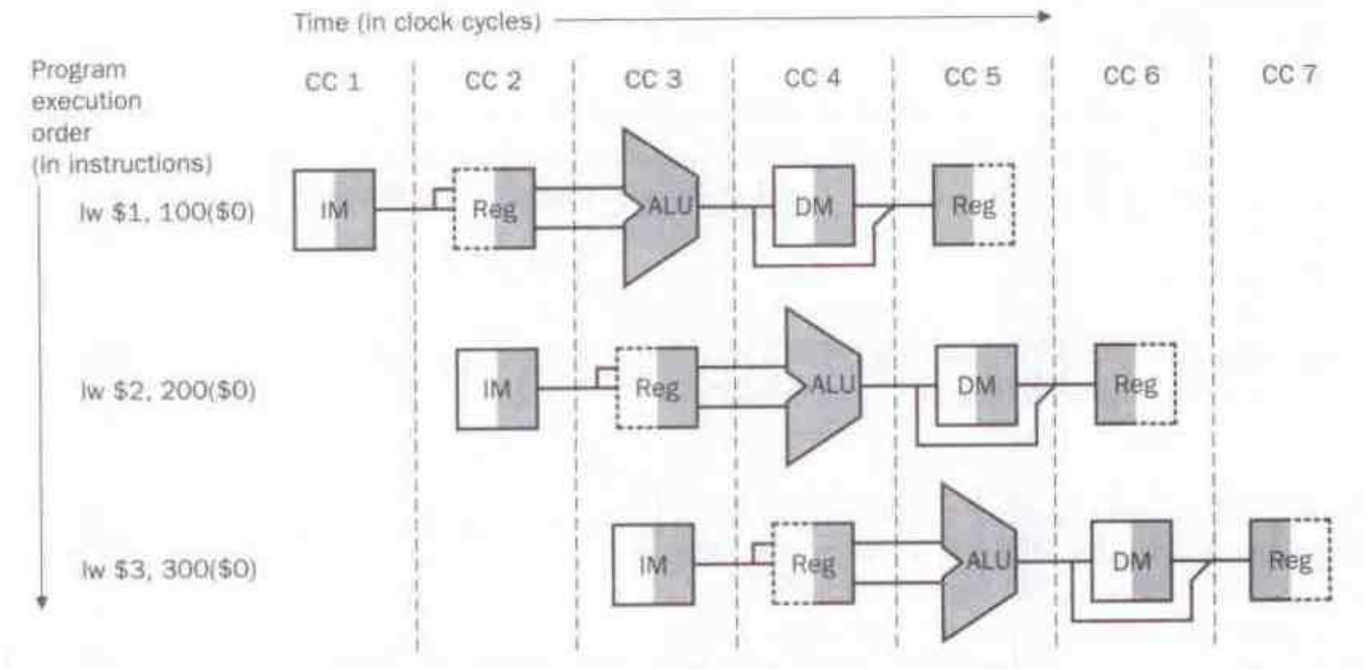
- Design of the data path
  - Working of a microprocessor
    - Instruction fetch
      - Instruction fetch unit
    - Instruction decoding
      - Decoding logic
    - Execution (include memory write operation)
      - ALU, register files, memory
  - Overall Data Path Design



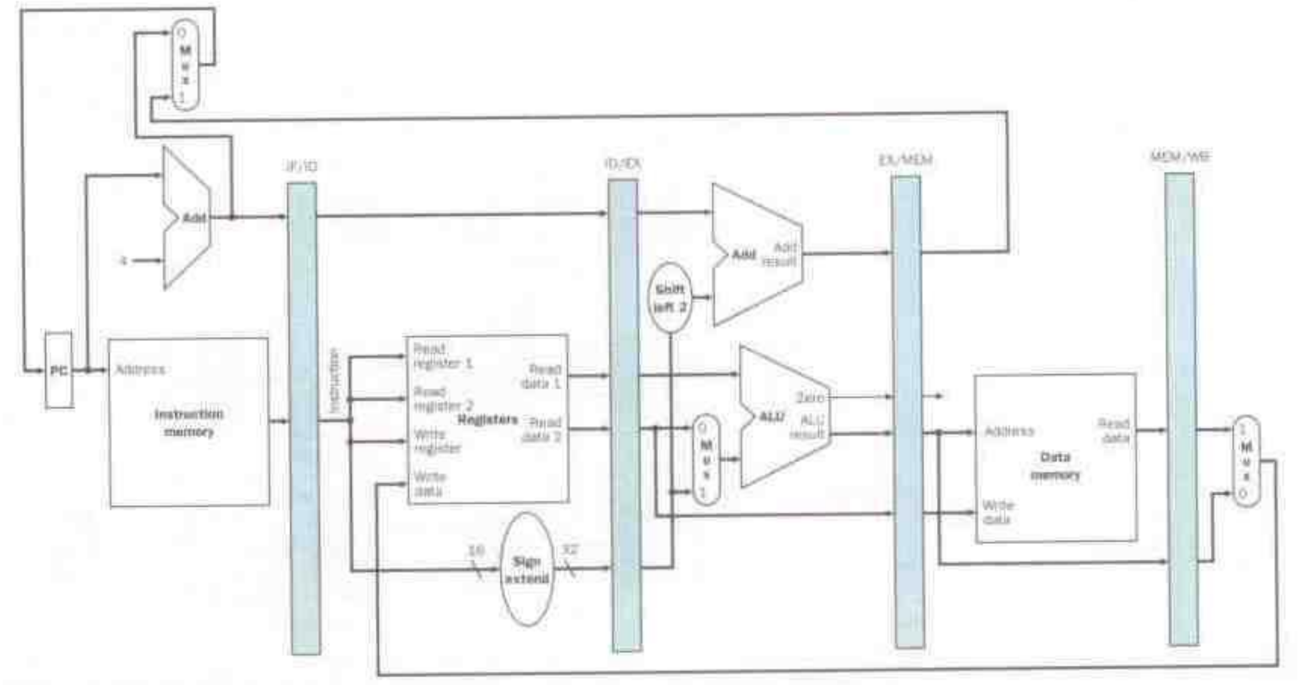
- Flow Chart for Instruction Processing



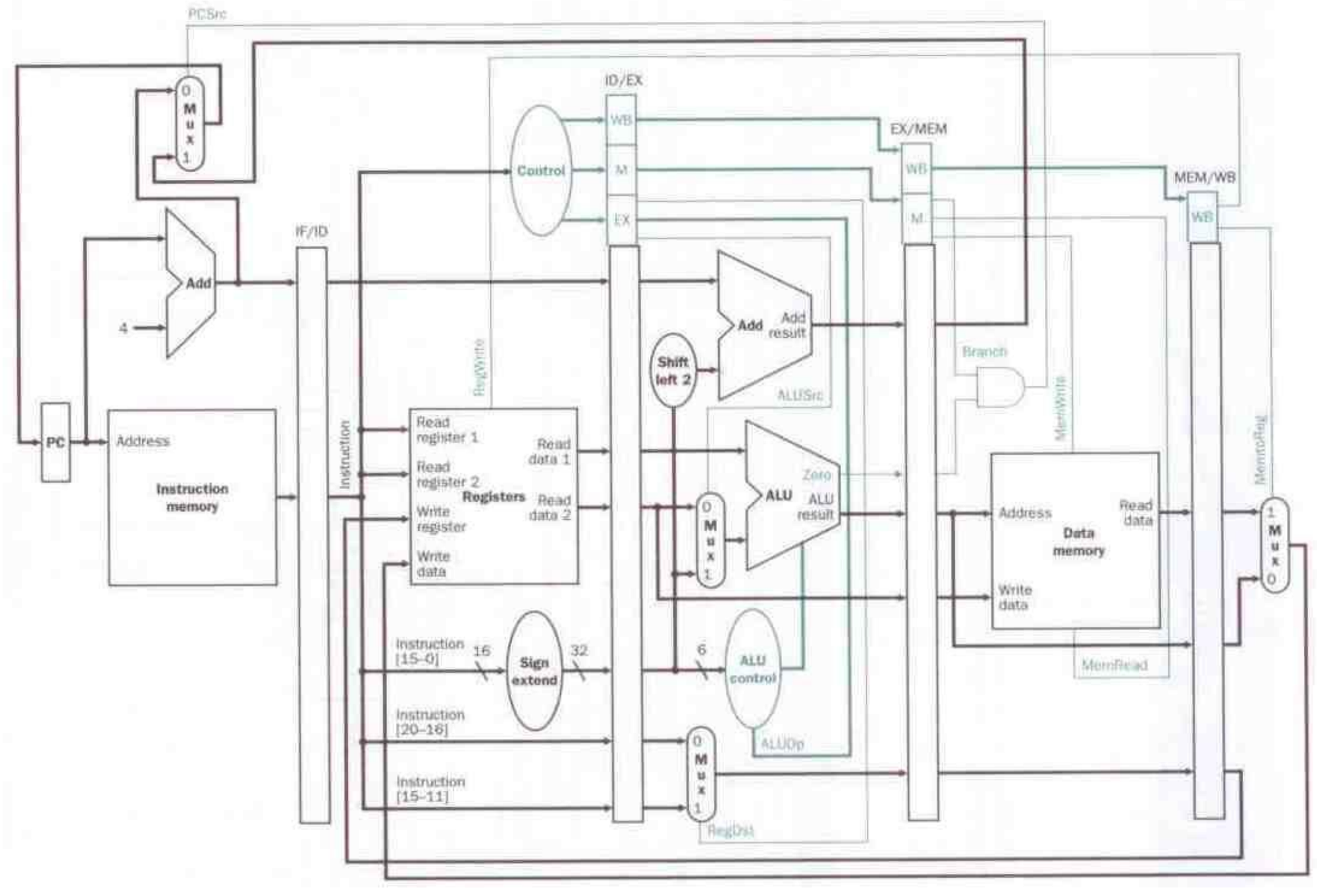
- Pipelining the design for efficiency



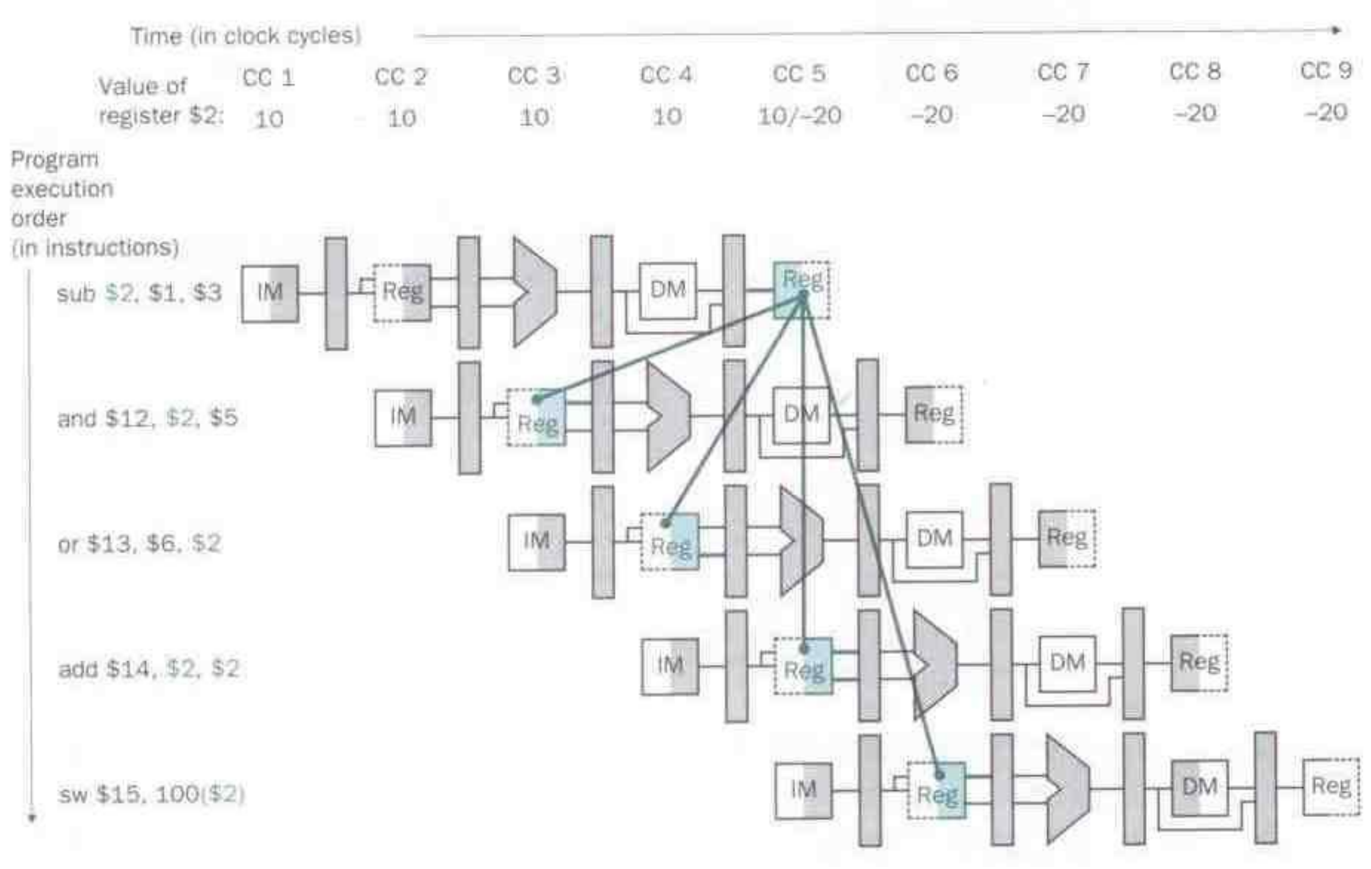
- Pipelined Data Path



- Pipelined Data Path with Control Signals



- Hazards in the pipelined design and data dependency
    - Data Hazard
    - Control Hazard (Branch Hazard)
    - Solutions
        - Data forwarding, Stall the pipeline


Time (in clock cycles)
Value of register $2:
CC 1 10
CC 2 10
CC 3 10
CC 4 10
CC 5 10/–20
CC 6 –20
CC 7 –20
CC 8 –20
CC 9 –20
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
IM
Reg
DM