2.5 Representing Instructions in Computer

- Instructions are encoded in binary
  - Called machine code
- MIPS instructions
  - Encoded as 32-bit instruction words

Representation of a Program

- High-level language
  - Level of abstraction closer to the problem domain
  - Provides productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

```
void swap (int v[], int k){
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
}
```

source file

compiler

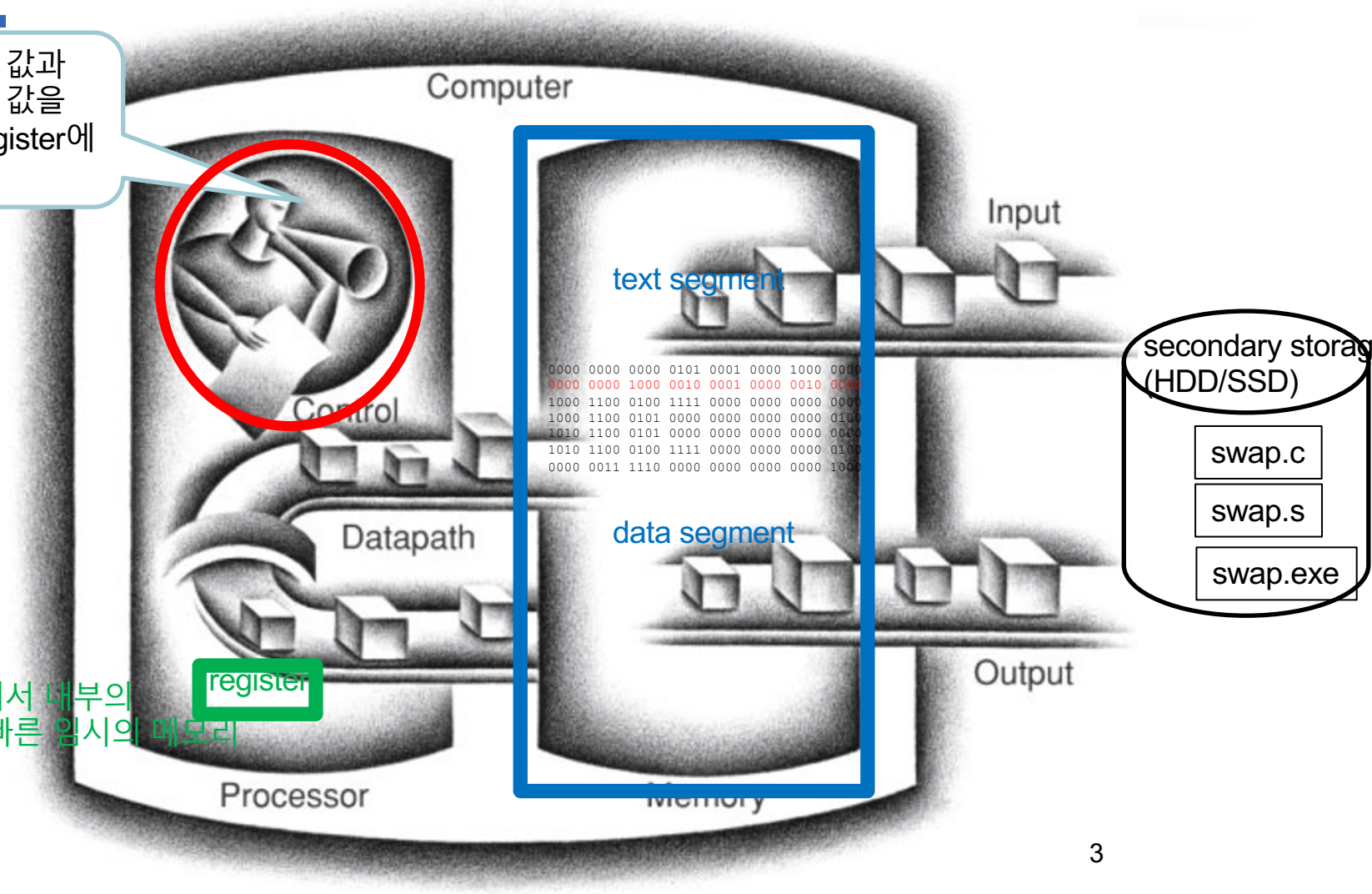
```
swap :   sll   $2, $5, 2
         add   $2, $4, $2
         lw    $15, 0($2)
         lw    $16, 4($2)
         sw    $16, 0($2)
         sw    $15, 4($2)
         jr    $31
```

assembler

executable file

```
0000 0000 0000 0101 0001 0000 1000 0000
0000 0000 1000 0010 0001 0000 0010 0000
1000 1100 0100 1111 0000 0000 0000 0000
1000 1100 0101 0000 0000 0000 0000 0100
1010 1100 0101 0000 0000 0000 0000 0000
1010 1100 0100 1111 0000 0000 0000 0100
0000 0011 1110 0000 0000 0000 0000 1000
```

4번 register의 값과
2번 register의 값을
더해서 2번 register에
써라.



프로세서 내부의
작고 빠른 임시의 메모리

QtSpim

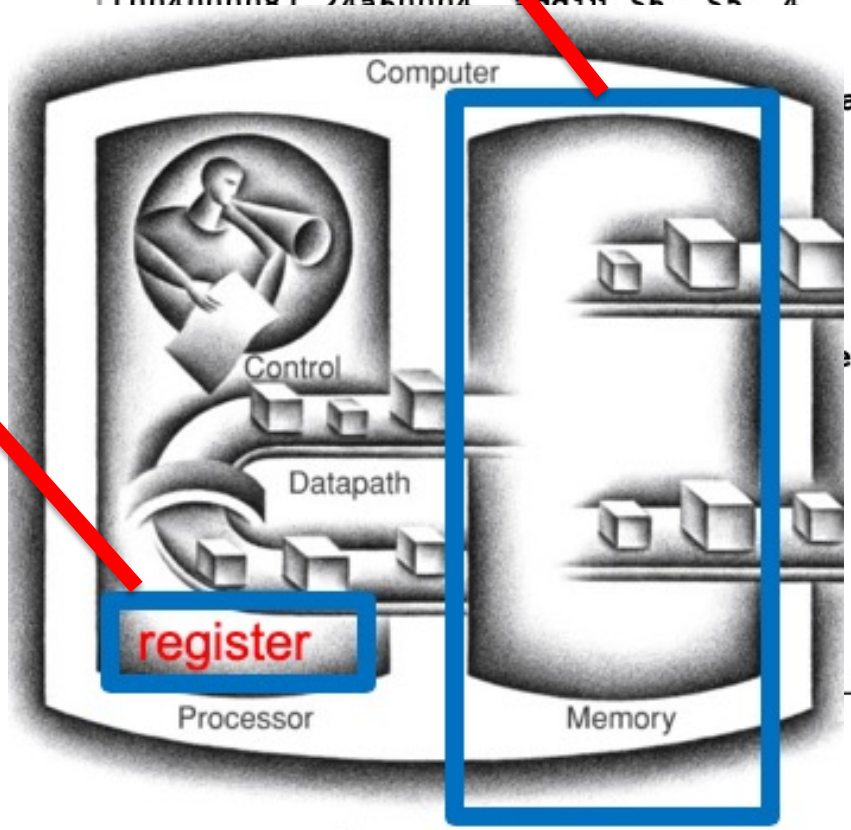
FP Regs | Int Regs [16] | Data | Text

Int Regs [16]

```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 0
R6  [a2] = 7ffffe
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
```

Text

```
                                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # arg
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # env
                                                ; 186: sll $v0 $a0 2
                                                ; 187: addu $a2 $a2 $v0
                                    ain]        ; 188: jal main
                                                ; 189: nop
                                                ; 191: li $v0 10
                                                ; 192: syscall # syscall 10
                                                ; 4: addi $5, $0, 10
                                                ; 5: addi $6, $0, 20
                                                ; 6: add $7, $5, $6

                                    ext Segment [80000000]..[80010000]
                                                ; 90: move $k1 $at # Save $a
                                                ; 92: sw $v0 s1 # Not re-ent
                                                ; 93: sw $a0 s2 # But we nee
                                                ; 95: mfc0 $k0 $13 # Cause r
                                                ; 96: srl $a0 $k0 2 # Extrac
                                                ; 97: andi $a0 $a0 0x1f
                                                ; 101: li $v0 4 # syscall 4
                                    _m1_]       ; 102: la $a0 __m1_
                                                ; 103: syscall
                                                ; 105: li $v0 1 # syscall 1
                                                ; 106: srl $a0 $k0 2 # Extra
                                                ; 107: andi $a0 $a0 0x1f
                                                ; 108: syscall
[800001b8] 0000000c  syscall
[800001bc] 34020004  ori $2, $0, 4              ; 110: li $v0 4 # syscall 4
[800001c0] 3344003c  andi $4, $26, 60           ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672             ; 112: lw $a0 __excp($a0)
```




```
SPIM Version 9.1.20 of August 29, 2017
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.
```

QtSpim

FP Regs | Int Regs [10] | Data | Text

Int Regs [10]

```
PC          = 0
EPC         = 0
Cause       = 0
BadVAddr    = 0
Status      = 805371664

HI          = 0
LO          = 0

R0  [r0]  = 0
R1  [at]  = 0
R2  [v0]  = 0
R3  [v1]  = 0
R4  [a0]  = 3
R5  [a1]  = 2147483100
R6  [a2]  = 2147483116
R7  [a3]  = 0
R8  [t0]  = 0
R9  [t1]  = 0
R10 [t2]  = 0
R11 [t3]  = 0
R12 [t4]  = 0
R13 [t5]  = 0
R14 [t6]  = 0
R15 [t7]  = 0
R16 [s0]  = 0
R17 [s1]  = 0
R18 [s2]  = 0
R19 [s3]  = 0
R20 [s4]  = 0
R21 [s5]  = 0
R22 [s6]  = 0
R23 [s7]  = 0
```

Text

```
                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) #
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscal
[00400024] 200b0007  addi $11, $0, 7            ; 4: addi $t3, $0, 7
[00400028] 200c0006  addi $12, $0, 6            ; 5: addi $t4, $0, 6
[0040002c] 016c6820  add $13, $11, $12          ; 6: add $t5, $t3, $t4
[00400030] 016c7022  sub $14, $11, $12          ; 7: sub $t6, $t3, $t4
[00400034] 200ffffa  addi $15, $0, -6           ; 8: addi $t7, $0, -6
[00400038] 016fc020  add $24, $11, $15          ; 9: add $t8, $t3, $t7

                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Sa
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not r
can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But w
registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Ca
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # R
[8000019c] 3084001f  andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4              ; 101: li $v0 4 # sysca
[800001a4] 3c049000  lui $4, -28672 [__m1_]     ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                    ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1              ; 105: li $v0 1 # sysca
[800001b0] 001a2082  srl $4, $26, 2             ; 106: srl $a0 $k0 2 #
```

MIPS R-format Instructions

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

- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount (00000 for now)
  - funct: function code (extends opcode)

R-format Example : add

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

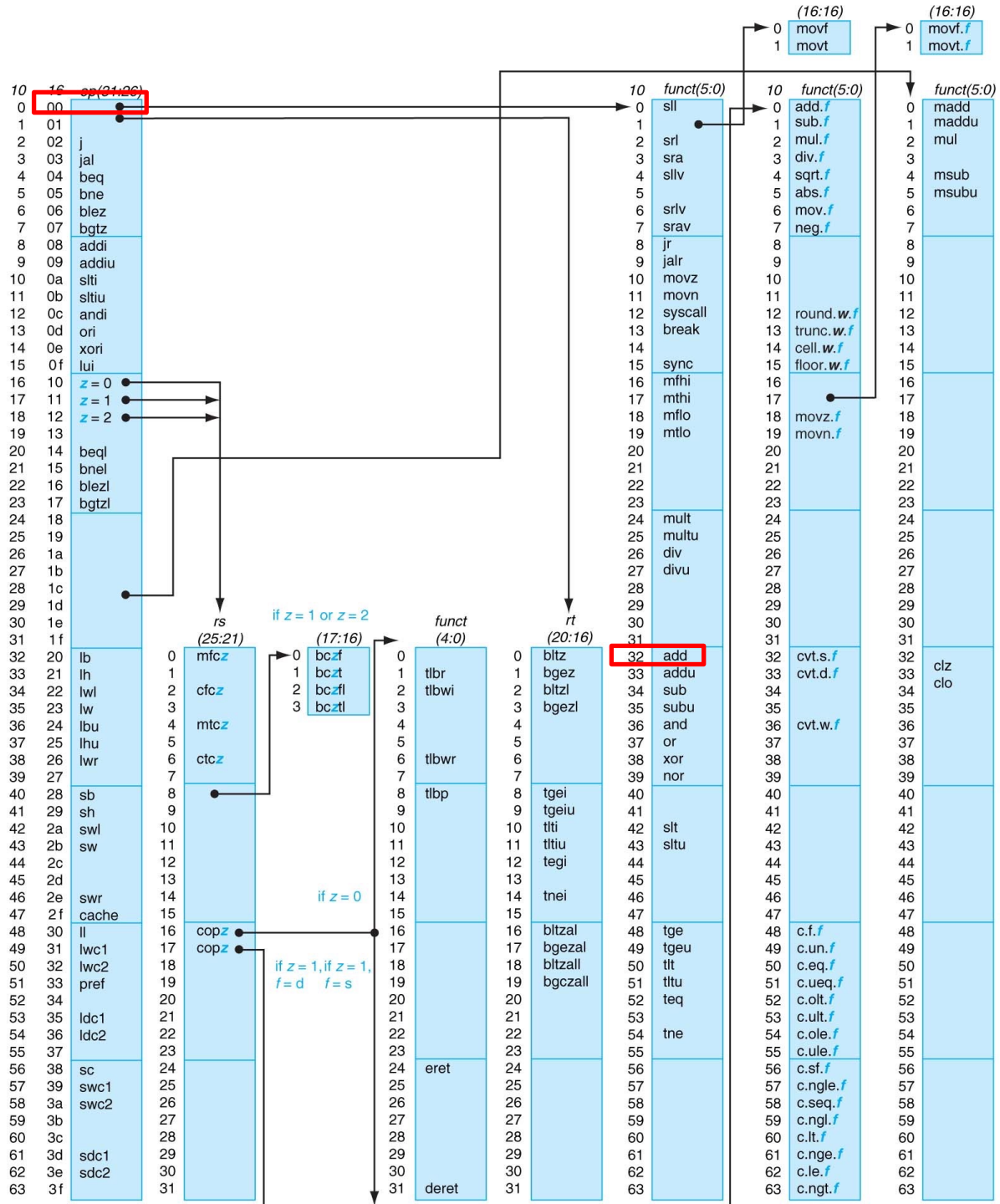
```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$$00000010001100100100000000100000_2 = 02324020_{16}$$

MIPS opcode map

- **Appendix A-50**



MIPS Reference Data

①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		$0 / 08_{hex}$
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		$0 / 25_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}

Pseudoinstructions follow roughly the same conventions, but omit instruction encoding information. For example:

Multiply (without overflow)

`mul rdest, rsrc1, src2` *pseudoinstruction*

In pseudoinstructions, `rdest` and `rsrc1` are registers and `src2` is either a register or an immediate value. In general, the assembler and SPIM translate a more general form of an instruction (e.g., `add $v1, $a0, 0x55`) to a specialized form (e.g., `addi $v1, $a0, 0x55`).

Arithmetic and Logical Instructions

Absolute value

`abs rdest, rsrc` *pseudoinstruction*

Put the absolute value of register `rsrc` in register `rdest`.

Addition (with overflow)

`add rd, rs, rt`

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

Disassemble the following MIPS instruction (기계어→어셈블리어)

- 0x00853022

= 0000 0000 1000 0101 0011 0000 0010 0010

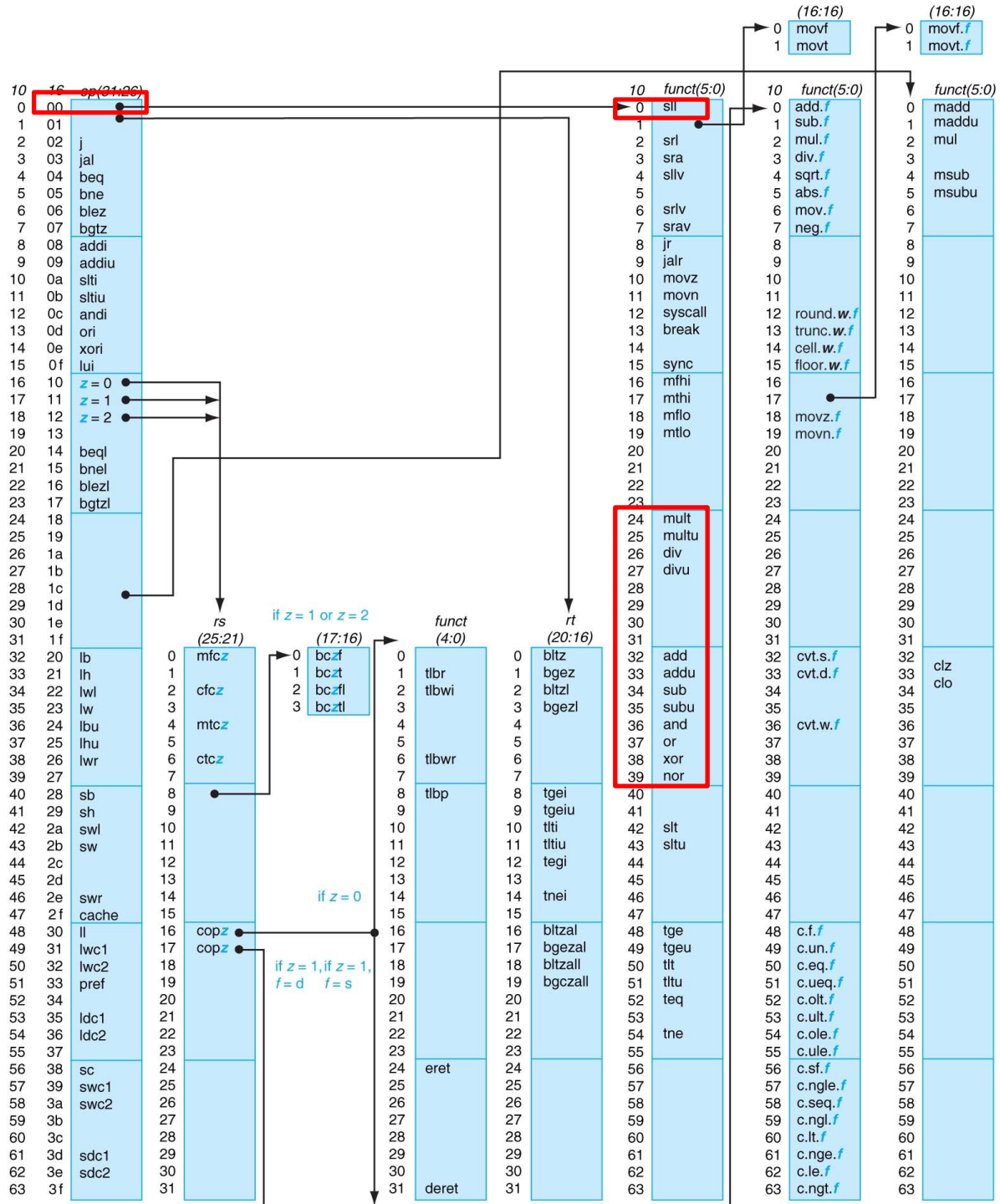
= 000000 00100 00101 00110 00000 100010

000000	00100	00101	00110	00000	100010
op	rs	rt	rd	shamt	funct

= `sub $6, $4, $5`

MIPS opcode map

- **Appendix A-50**



R-format instructions

- **add / addu**
- **sub / subu**
- **or / and / nor**
- **sll / srl**

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

```
sll    $10, $16, 4
```

000000 00000 10000 01010 00100 000000

0000 0000 0001 0000 0101 0001 0000 0000

= 0x00105100

FP Regs | Int Regs [16] | Data | Text

Int Regs [16]

```
PC        = 0
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 0
R6  [a2] = 7ffffe3c
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
```

Text

```
                        User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscall 10 (exit)
[00400024] 02324020  add $8, $17, $18           ; 4: add $t0, $s1, $s2 # 0x2324020
[00400028] 00853022  sub $6, $4, $5             ; 5: sub $6, $4, $5 # 0x00853022
[0040002c] 00105100  sll $10, $16, 4            ; 6: sll $10, $16, 4 # 0x00105100

                       Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not re-entrant a
we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But we need to u
these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Cause registe
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # Extract
ExcCode Field
```

```
Memory and registers cleared

SPIM Version 9.1.20 of August 29, 2017
Copyright 1990-2017 by James Larus.
All Rights Reserved.
```

QtSpim

FP Regs | Int Regs [10] | Data | Text

Int Regs [10]

```
PC          = 0
EPC         = 0
Cause       = 0
BadVAddr    = 0
Status      = 805371664

HI          = 0
LO          = 0

R0  [r0]  = 0
R1  [at]  = 0
R2  [v0]  = 0
R3  [v1]  = 0
R4  [a0]  = 3
R5  [a1]  = 2147483100
R6  [a2]  = 2147483116
R7  [a3]  = 0
R8  [t0]  = 0
R9  [t1]  = 0
R10 [t2]  = 0
R11 [t3]  = 0
R12 [t4]  = 0
R13 [t5]  = 0
R14 [t6]  = 0
R15 [t7]  = 0
R16 [s0]  = 0
R17 [s1]  = 0
R18 [s2]  = 0
R19 [s3]  = 0
R20 [s4]  = 0
R21 [s5]  = 0
R22 [s6]  = 0
R23 [s7]  = 0
```

Text

```
                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) #
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscal
[00400024] 200b0007  addi $11, $0, 7            ; 4: addi $t3, $0, 7
[00400028] 200c0006  addi $12, $0, 6            ; 5: addi $t4, $0, 6
[0040002c] 016c6820  add $13, $11, $12          ; 6: add $t5, $t3, $t4
[00400030] 016c7022  sub $14, $11, $12          ; 7: sub $t6, $t3, $t4
[00400034] 200ffffa  addi $15, $0, -6           ; 8: addi $t7, $0, -6
[00400038] 016fc020  add $24, $11, $15          ; 9: add $t8, $t3, $t7

                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Sa
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not r
can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But w
registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Ca
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # R
[8000019c] 3084001f  andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4              ; 101: li $v0 4 # sysca
[800001a4] 3c049000  lui $4, -28672 [__m1_]     ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                    ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1              ; 105: li $v0 1 # sysca
[800001b0] 001a2082  srl $4, $26, 2             ; 106: srl $a0 $k0 2 #
```

Disassemble

0x016C6820

= 0000 0001 0110 1100 0110 1000 0010 0000

000000 01011 01100 01101 00000 100000

```
add $13, $11, $12
```

R-format Example : add

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

```
add $t5, $t3, $t4
```

special	$t3	$t4	$t5	0	add
0	11	12	13	0	32
000000	01011	01100	01101	00000	100000

$0000\ 0001\ 0110\ 1100\ 0110\ 1000\ 0010\ 0000_2 = 016c6820_{16}$