

Lecture Handout

Computer Architecture

Lecture No. 4

Reading Material

Vincent P. Heuring & Harry F. Jordan
Computer Systems Design and Architecture

Chapter 2
2.3, 2.4, slides

Summary

- 1) Introduction to ISA and instruction formats
- 2) Coding examples and Hand assembly

An example computer: the SRC: “simple RISC computer”

An example machine is introduced here to facilitate our understanding of various design steps and concepts in computer architecture. This example machine is quite simple, and leaves out a lot of details of a real machine, yet it is complex enough to illustrate the fundamentals.

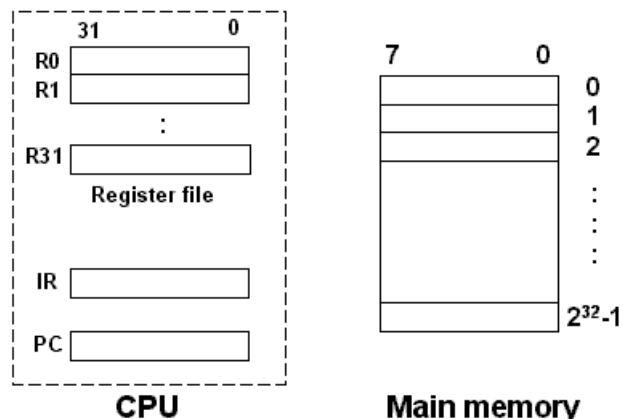
SRC Introduction

Attributes of the SRC

- The SRC contains 32 General Purpose Registers: R0, R1, ..., R31; each register is of size 32-bits.
- Two special purpose registers are included: Program Counter (PC) and Instruction Register (IR)
- Memory word size is 32 bits
- Memory space size is 2^{32} bytes
- Memory organization is $2^{32} \times 8$ bits, this means that the memory is byte aligned
- Memory is accessed in 32 bit words (i.e., 4 byte chunks)
- Big-endian byte storage is used

Programmer’s View of the SRC

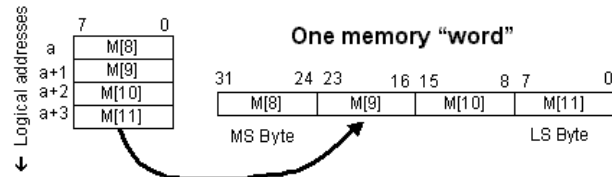
The figure below shows the attributes of the SRC; the 32 ,32-bit registers that are a part of the CPU, the two additional CPU registers (PC & IR), and the main memory which is 2^{32} 1-byte cells.



SRC Notation

We examine the notation used for the SRC with the help of some examples.

- R[3] means contents of register 3 (R for register)
- M[8] means contents of memory location 8 (M for memory)
- A memory word at address 8 is defined as the 32 bits at address 8,9,10 and 11 in the memory. This is shown in the figure below.
- A special notation for 32-bit memory words is



$M[8]<31...0>:=M[8]\odot M[9]\odot M[10]\odot M[11]$

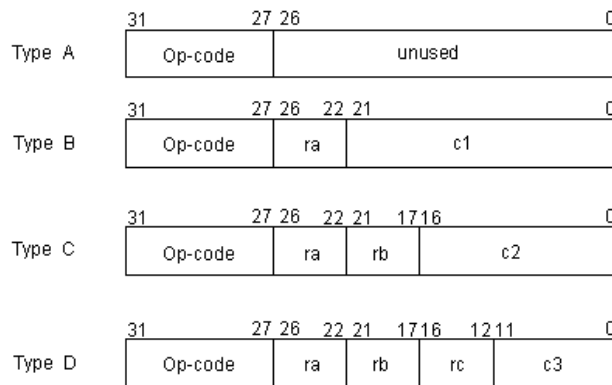
\odot is used for concatenation.

Some more SRC Attributes

- All instructions are 32 bits long (i.e., instruction size is 1 word)
- All ALU instructions have three operands
- The only way to access memory is through load and store operations
- Only a few addressing modes are supported

SRC: Instruction Formats

Four types of instructions are supported by the SRC. Their representation is given in the following figure. Before discussing these instruction types in detail, we take a look at the encoding of general-purpose registers (the ra, rb and rc fields).



Encoding of the General Purpose Registers

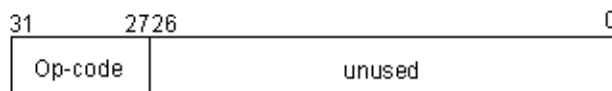
The encoding for the general purpose registers is shown in the following table; it will be used in place of ra, rb and rc in the instruction formats shown above. Note that this is a simple 5 bit encoding. ra, rb and rc are names of fields used as “place-holders”, and can represent any one of these 32 registers. An exception is rb = 0; it does not mean the register R0, rather it means no operand. This will be explained in the following discussion.

Register	Code	Register	Code	Register	Code	Register	Code
R0	00000	R8	01000	R16	10000	R24	11000
R1	00001	R9	01001	R17	10001	R25	11001
R2	00010	R10	01010	R18	10010	R26	11010
R3	00011	R11	01011	R19	10011	R27	11011
R4	00100	R12	01100	R20	10100	R28	11100
R5	00101	R13	01101	R21	10101	R29	11101
R6	00110	R14	01110	R22	10110	R30	11110
R7	00111	R15	01111	R23	10111	R31	11111

Type A

Type A is used for only two instructions:

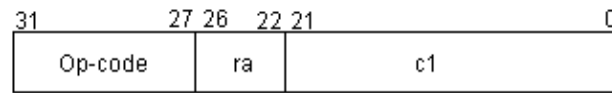
- No operation or nop, for which the op-code = 0. This is useful in pipelining
- Stop operation stop, the op-code is 31 for this instruction.



Both of these instructions do not need an operand (are 0-operand instructions).

Type B

Type B format includes three instructions; all three use relative addressing mode. These are

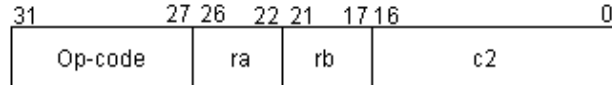


- The ldr instruction, used to load register from memory using a relative address. (op-code = 2).
 - Example:
ldr R3, 56
This instruction will load the register R3 with the contents of the memory location M [PC+56]
- The lar instruction, for loading a register with relative address (op-code = 6)
 - Example:
lar R3, 56
This instruction will load the register R3 with the relative address itself (PC+56).
- The str is used to store register to memory using relative address (op-code = 4)
 - Example:
str R8, 34
This instruction will store the register R8 contents to the memory location M [PC+34]

The effective address is computed at run-time by adding a constant to the PC. This makes the instructions 're-locatable'.

Type C

Type C format has three load/store instructions, plus three ALU instructions. These load/ store instructions are



- ld, the load register from memory instruction (op-code = 1)
 - Example 1:
ld R3, 56
This instruction will load the register R3 with the contents of the memory location M [56]; the rb field is 0 in this instruction, i.e., it is not used. This is an example of direct addressing mode.
 - Example 2:
ld R3, 56(R5)
The contents of the memory location M [56+R [5]] are loaded to the register R3; the rb field \neq 0. This is an instance of indexed addressing mode.
- la is the instruction to load a register with an immediate data value (which can be an address) (op-code = 5)
 - Example1:
la R3, 56
The register R3 is loaded with the immediate value 56. This is an instance of immediate addressing mode.
 - Example 2:
la R3, 56(R5)

The register R3 is loaded with the indexed address $56 + R[5]$. This is an example of indexed addressing mode.

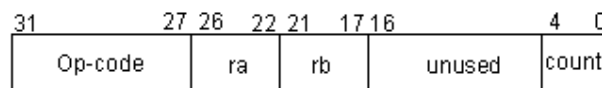
- The `st` instruction is used to store register contents to memory (op-code = 3)
 - Example 1:
`st R8, 34`
 This is the direct addressing mode; the contents of register R8 ($R[8]$) are stored to the memory location $M[34]$
 - Example 2:
`st R8, 34(R6)`
 An instance of indexed addressing mode, $M[34 + R[6]]$ stores the contents of $R8(R[8])$

The ALU instructions are

- `addi`, immediate 2's complement addition (op-code = 13)
 - Example:
`addi R3, R4, 56`
 $R[3] \leftarrow R[4] + 56$ (*rb field = R4*)
- `andi`, the instruction to obtain immediate logical AND, (op-code = 21)
 - Example:
`andi R3, R4, 56`
 R3 is loaded with the immediate logical AND of the contents of register R4 and 56(constant value)
- `ori`, the instruction to obtain immediate logical OR (op-code = 23)
 - Example:
`ori R3, R4, 56`
 R3 is loaded with the immediate logical OR of the contents of register R4 and 56(constant value)

Note:

1. Since the constant `c2` field is 17 bits,
 - For direct addressing mode, only the first 2^{16} bytes of memory can be accessed (or the last 2^{16} bytes if `c2` is negative)
 - In case of the `la` instruction, only constants with magnitudes less than $\pm 2^{16}$ can be loaded
 - During address calculation using `c2`, sign extension to 32 bits must be performed before the addition
2. Type C instructions, with some modifications, may also be used for shift instructions. Note the modification in the following figure.



The four shift instructions are

- `shr` is the instruction used to shift the bits right by using value in (5-bit) `c3` field(shift count) (op-code = 26)
 - Example:
`shr R3, R4, 7`
 shift R4 right 7 times in to R3 and shifts zeros in from the left as the value is shifted right. Immediate addressing mode is used.
- `shra`, arithmetic shift right by using value in `c3` field (op-code = 27)
 - Example:

shra R3, R4, 7

This instruction has the effect of shift R4 right 7 times in to R3 and copies the msb into the word on left as contents are shifted right. Immediate addressing mode is used.

- The shl instruction is for shift left by using value in (5-bit) c3 field (op-code = 28)
 - Example:

shl R8, R5, 6

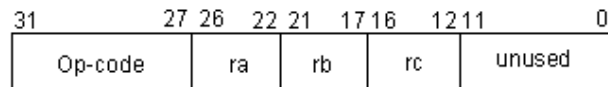
shift R5 left 6 times in to R8 and shifts zeros in from the right as the value is shifted left. Immediate addressing mode is used.
- shc, shift left circular by using value in c3 field (op-code = 29)
 - Example:

shc R3, R4, 3

shift R4 circular 3 times in to R3 and copies the value shifted out of the register on the left is placed back into the register on the right. Immediate addressing mode is used.

Type D

Type D includes four ALU instructions, four register based shift instructions, two logical instructions and two branch instructions.



The four ALU instructions are given below

- add, the instruction for 2's complement register addition (op-code = 12)
 - Example:

add R3, R5, R6

result of 2's complement addition $R[5] + R[6]$ is stored in R3. Register addressing mode is used.
- sub, the instruction for 2's complement register subtraction (op-code = 14)
 - Example:

sub R3, R5, R6

R3 will store the 2's complement subtraction, $R[5] - R[6]$. Register addressing mode is used.
- and, the instruction for logical AND operation between registers (op-code = 20)
 - Example:

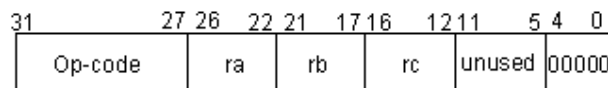
and R8, R3, R4

R8 will store the logical AND of registers R3 and R4. Register addressing mode is used.
- or, the instruction for logical OR operation between registers (op-code = 22)
 - Example:

or R8, R3, R4

R8 is loaded with the value $R[3] \vee R[4]$, the logical OR of registers R3 and R4. Register addressing mode is used.

The four register based shift instructions use register addressing mode. These use a modified form of type D, as shown in figure



- shr, shift right by using value in register rc (op-code = 26)
 - Example:

shr R3, R4, R5

This instruction will shift R4 right in to R3 using number in R5

- shra, the arithmetic shift right by using register rc (op-code = 27)

- Example:

shra R3, R4, R5

A shift of R4 right using R5, and the result is stored in R3

- shl is shift left by using register rc (op-code = 28)

- Example:

shl R8, R5, R6

The instruction shifts R5 left in to R8 using number in R6

- shc, shifts left circular by using register rc (op-code = 29)

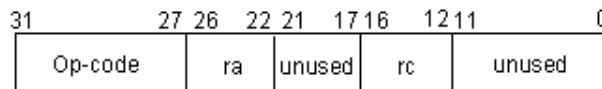
- Example:

shc R3, R4, R6

This instruction will shift R4 circular in to R3 using value in R6

The two logical instructions also use a modified form of the Type D, and are the following.

- neg stores the 2's complement of register rc in ra (op-code = 15)



- Example:

neg R3, R4

Negates (obtains 2's complement) of R4 and stores in R3. 2-address format and register addressing mode is used.

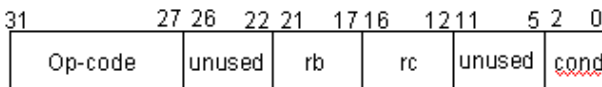
- not stores the 1's complement of register rc in ra (op-code = 24)

- Example:

not R3, R4

Logically inverts R4 and stores in R3. 2-address format with register addressing mode is used.

Type D has two-branch instruction, modified forms of type D.



- br , the instruction to branch to address in rb depending on the condition in rc. There are five possible conditions, explained through examples. (op-code = 8). All branch instructions use register-addressing mode.

- Example 1:

brzr R3, R4

Branch to address in R3 (if R4 == 0)

- Example 2:

brnz R3, R4

Branch to address in R3 (if R4 ≠ 0)

- Example 3:

brpl R3, R4

Branch to address in R3 (if R4 ≥ 0)

- Example 4:

brmi R3, R4

Branch to address in R3 (if R4 < 0)

- Example 5:

Advanced Computer Architecture-CS501

br R3, R4

Branch to address in R3 (unconditional)

- Brl the instruction to branch to address in rb depending on condition in rc. Additionally, it copies the PC in to ra before branching (op-code = 9)

- Example 1:

brl_z R1,R3, R4

R1 will store the contents of PC, then branch to address in R3 (if R4 == 0)

- Example 2:

brlnz R1,R3, R4

R1 stores the contents of PC, then a branch is taken, to address in R3 (if)

- Example 3:

brlpl R1,R3, R4

R1 will store PC, then branch to address in R3 (if $R4 \geq 0$)

- Example 4:

brlmi R1,R3, R4

R1 will store PC and then branch to address in R3 (if R4 < 0)

- Example 5:

brl R1,R3, R4

R1 will store PC, then it will ALWAYS branch to address in R3

- Example 6:

$$\text{brlnv } R1, R3, R4$$

R1 just stores the contents of PC but a branch is not taken (NEVER BRANCH)

Mnemonic	e3<2..0>	Branch Condition
brlnv	000	Link but never branch ⁺
br, brl	001	Unconditional branch
brzr, brlzl	010	Branch if rc is zero
brnz, brnlz	011	Branch if rc is not zero
brpl, brlpl	100	Branch if rc is positive
brmi, brlmi	101	Branch if rc is negative

In the modified type D instructions for branch, the bits <2..0> are used for specifying the condition; these condition codes are shown in the table.

The SRC Instruction Summary

The instructions implemented by the SRC are listed, grouped on functionality basis.

Functional Groups of Instructions

Logic		Opcode			
Shift right by count	shr	1	1	0	1
Shift right by count in a register	shr	1	1	0	1
AShift right by count	shra	1	1	0	1
AShift right by count in a register	shra	1	1	0	1
Shift left by count	shl	1	1	1	0
Shift left by count in a register	shl	1	1	1	0
Shift circ. by count	shc	1	1	1	0
Shift circ. by count in a register	shc	1	1	1	0

Arithmetic							Opcode				
2's complement addition						add	0	1	1	0	0
Immediate 2's complement addn.						addi	0	1	1	0	1
2's c	Mnemonic	31	30	29	28	27	la	0	0	1	0
Neg							lar	0	0	1	1
Log							ld	0	0	0	0
Imm							ldr	0	0	0	1
Log	add	0	1	1	0	0	neg	0	1	1	1
Imm	addi	0	1	1	0	1	...	0	0	0	0
Not											

2's	Mnemonic	31	30	29	28	27
Neg						
Log						
Imm		Opcode				
Log	add	0	1	1	0	0
Imm	addi	0	1	1	0	1
Not						

la	0	0	1	0	1
lar	0	0	1	1	0
ld	0	0	0	0	1
ldr	0	0	0	1	0
neg	0	1	1	1	1
neg	0	0	0	0	0

Con:	br	0	1	0	0	0
	brl	0	1	0	0	1
No o	brlmi	0	1	0	0	1
Half	brlnv	0	1	0	0	1
	brlnz	0	1	0	0	1
Data	brlpl	0	1	0	0	1
	brlzs	0	1	0	0	1
Load	brmi	0	1	0	0	0
Load	brnv	0	1	0	0	0
Store	brnz	0	1	0	0	0
Store	brpl	0	1	0	0	0
Load	brzs	0	1	0	0	0
Load	brzs	0	1	0	0	0

not	1	1	0	0	0				
or	1	0	1	1	0				
ori	1	0	1	1	1	Opcode			
sbc	1	1	1	0	1	0	1	0	0
sbc	1	1	1	0	1	0	1	0	0
sbl	1	1	1	0	0	0	1	0	0
sbl	1	1	1	0	0	0	1	0	0
shr	1	1	0	1	0	0	1	0	0
shr	1	1	0	1	0	0	1	0	0
shra	1	1	0	1	1				
shra	1	1	0	1	1	0	1	0	0
st	0	0	0	1	1	0	1	0	0
stop	1	1	1	1	1	0	1	0	0
str	0	0	1	0	0	0	1	0	0
sub	0	1	1	1	0	0	1	0	0

Alphabetical Listing based on SRC Mnemonics

Notice that the op code field for all br instructions is the same. The difference is in the condition code field, which is in effect, an op code extension.

Examples

Some examples are studied in this section to enhance the student's understanding of the SRC.

Example 1: Expression Evaluation

Write an SRC assembly language program to evaluate the expression:

$$z = 4(a + b) - 16(c + 58)$$

Your code should not change the source operands.

Solution A: Notice that the SRC does not have a multiply instruction. We will make use of the fact that multiplication with powers of 2 can be achieved by repeated shift left operations. A possible solution is give below:

```
ld R1, c           ; c is a label used for a memory location
addi R3, R1, 58     ; R3 contains (c+58)
shl R7, R3, 4       ; R7 contains 16(c+58)
ld R4, a
ld R5, b
add R6, R4, R5      ; R6 contains (a+b)
shl R8, R6, 2       ; R8 contains 4(a+b)
sub R9, R7, R8      ; the result is in R9
st R9, z           ; store the result in memory location z
```

Note:

The memory labels a, b, c and z can be defined by using assembler directives like .dw or .db, etc. in the source file.

A semicolon ';' is used for comments in assembly language.

Solution B:

We may solve the problem by assuming that a multiply instruction, similar to the add instruction, exists in the instruction set of the SRC. The shl instruction will be replaced by the mul instruction as given below.

```
ld R1, c           ; c is a label used for a memory location
addi R3, R1, 58     ; R3 contains (c+58)
mul R7, R3, 4       ; R7 contains 16(c+58)
ld R4, a
ld R5, b
add R6, R4, R5      ; R6 contains (a+b)
mul R8, R6, 2       ; R8 contains 4(a+b)
sub R9, R7, R8      ; the result is in R9
st R9, z           ; store the result in memory location z
```

Note:

The memory labels a, b, c and z can be defined by using assembler directives like .dw or .db, etc. in the source file.

Solution C:

We can perform multiplication with a multiplier that is not a power of 2 by doing addition in a loop. The number of times the loop will execute will be equal to the multiplier.

Example 2: Hand Assembly

Convert the given SRC assembly language program in to an equivalent SRC machine language program.

```
ld R1, c           ; c is a label used for a memory location
addi R3, R1, 58     ; R3 contains (c+58)
```


Advanced Computer Architecture-CS501

```
shl R7, R3, 4          ; R7 contains 16(c+58)
ld R4, a
ld R5, b
add R6, R4, R5          ; R6 contains (a+b)
shl R8, R6, 2           ; R8 contains 4(a+b)
sub R9, R7, R8           ; the result is in R9
st R9, z                ; store the result in memory location z
```

Note:

This program uses memory labels a,b,c and z. We need to define them for the assembler by using assembler directives like .dw or .equ etc. in the source file.

Assembler Directives

Assembler directives, also called pseudo op-codes, are commands to the assembler to direct the assembly process. The directives may be slightly different for different assemblers. All the necessary directives are available with most assemblers. We explain the directives as we encounter them. More information on assemblers can be looked up in the assembler user manuals.

Source program with directives

```
                .ORG    200    ; start the next line at address 200
a:              .DW      1      ; reserve one word for the label a in the memory
b:              .DW      1      ; reserve a word for b, this will be at address 204
c:              .DW      1      ; reserve a word for c, will be at address 208
z:              .DW      1      ; reserve one word for the result
                .ORG    400    ; start the code at address 400
```

; all numbers are in decimal unless otherwise stated

```
ld R1, c        ; c is a label used for a memory location
```

```
addi R3, R1, 58 ; R3 contains (c+58)
```

```
shl R7, R3, 4    ; R7 contains 16(c+58)
```

```
ld R4, a
```

```
ld R5, b
```

```
add R6, R4, R5    ; R6 contains (a+b)
```

```
shl R8, R6, 2     ; R8 contains 4(a+b)
```

```
sub R9, R7, R8    ; the result is in R9
```

```
st R9, z          ; store the result in memory location z
```

This is the way an assembly program will appear in the source file. Most assemblers require that the file be saved with an .asm extension.

Solution:

Observe the first line of the program

.ORG 200 ; start the next line at address 200

This is a directive to let the following code/ variables ‘originate’ at the specified address of the memory, 200 in this case.

Variable statements, and another .ORG directive follow the .ORG directive.

```
a:              .DW      1      ; reserve one word for the label a in the memory
b:              .DW      1      ; reserve a word for b, this will be at address 204
c:              .DW      1      ; reserve a word for c, will be at address 208
z:              .DW      1      ; reserve one word for the result
                .ORG    400    ; start the code at address 400
```

We conclude the following from the above statements:

Label	Address	Value
a	200	unknown
b	204	unknown
c	208	unknown
z	212	unknown

Advanced Computer Architecture-CS501

The code starts at address 400 and each instruction takes 32 bits in the memory. The memory map for the program is shown in given table.

Memory Map for the SRC example program

Memory Address	Memory Contents
200	unknown
204	unknown
208	unknown
212	unknown
...	...
400	ld R1, c
404	addi R3, R1, 58
408	shl R7, R3, 4
412	ld R4, a
416	ld R5, b
420	add R6, R4, R5
424	shl R8, R6, 2
428	sub R9, R7, R8
432	st R9, z

We have to convert these instructions to machine language. Let us start with the first instruction:

ld R1, c

Notice that this is a type C instruction with the rb field missing.

1. We pick the op-code for this load instruction from the SRC instruction tables given in the SRC instruction summary section. The op-code for the load register 'ld' instruction is 00001.
2. Next we pick the register code corresponding to register R1 from the register table (given in the section 'encoding of general purpose registers'). The register code for R1 is 00001.
3. The rb field is missing, so we place zeros in the field: 00000
4. The value of c is provided by the assembler, and should be converted to 17 bits. As c has been assigned the memory address 208, the binary value to be encoded is 00000 0000 1101 0000.
5. So the instruction ld R1, c is 00001 00001 00000 00000 0000 1101 0000 in the machine language.
6. The hexadecimal representation of this instruction is 0 8 4 0 0 0 D 0 h.

We can update the memory map with these values.

We consider the next instruction,

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0h
404	addi R3, R1, 58	
408	shl R7, R3, 4	
412		
416		
420		
424		
428		
432		

Memory Address	Memory Contents	Hexadecimal Memory Contents
...
400	ld R1, c	084000D0h
404	addi R3, R1, 58	68C2003Ah
408	shl R7, R3, 4	
412	ld R4, a	
416	ld R5, b	
420	add R6, R4, R5	
424	shl R8, R6, 2	
428	sub R9, R7, R8	
432	st R9, z	

Advanced Computer Architecture-CS501

addi R3, R1, 58.

Notice that this is a type C instruction.

1. We pick the op-code for the instruction addi from the SRC instruction table. It is 01101
2. We pick the register codes for the registers R3 and R1, these codes are 00011 and 00001 respectively
3. For the immediate data, 58, we use the binary value, 00000 0000 0011 1010
4. So the complete instruction becomes: 01101 00011 00001 00000 0000 0011 1010
5. The hexadecimal representation of the instruction is 6 8 C 2 0 0 3 A h

We update the memory map, as shown in table.

The next instruction is **shl R7,R3, 4**, at address 408. Again, this is a type C instruction.

1. The op-code for the instruction shl is picked from the SRC instruction table. It is 11100
2. The register codes for the registers R7 and R3 from the register table are 00111 and 00011 respectively
3. For the immediate data, 4, the corresponding binary value 00000 0000 0000 0100 is used.
4. We can place these codes in accordance with the type C instruction format to obtain the complete instruction: 11100 00111 00011 00000 0000 0000 0100
5. The hexadecimal representation of the instruction is E1C60004

The memory map is updated, as shown in table.

The next instruction, **ld R4, a**, is also a type C instruction. Rb field is missing in this instruction. To obtain the machine equivalent, we follow the steps given below.

1. The op-code of the load instruction 'ld' is 00001
2. The register code corresponding to the register R4 is obtained from the register table, and it is 00100
3. As the 5 bit rb field is missing, we can encode zeros in its place: 00000
4. The value of a is provided by the assembler, and is converted to 17 bits. It has been assigned the memory address 200, the binary equivalent of which is: 00000 0000 1100 1000
5. The complete instruction becomes: 00001 00100 00000 00000 0000 1100 1000
6. The hexadecimal equivalent of the instruction is 0 9 0 0 0 0 C 8 h

Memory map can be updated with this value.

The next instruction is also a load type C instruction, with the rb field missing.

ld R5, b

The machine language conversion steps are

1. The op-code of the load instruction is obtained from the SRC instruction table; it is 00001
2. The register code for R5, obtained from the register table, is 00101

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	
416	ld R5, b	
420	add R6, R4, R5	
424	shl R8, R6, 2	
428	sub R9, R7, R8	
432	st R9, z	

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	090000C8 h
416	ld R5, b	
420	add R6, R4, R5	
424	shl R8, R6, 2	
428	sub R9, R7, R8	
432	st R9, z	

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	090000C8 h
416	ld R5, b	094000CC h
420	add R6, R4, R5	
424	shl R8, R6, 2	
428	sub R9, R7, R8	
432	st R9, z	

Advanced Computer Architecture-CS501

- Again, the 5 bit rb field is missing. We encode zeros in its place: 00000
- The value of label b is provided by the assembler, and should be converted to 17 bits. It has been assigned the memory address 204, so the binary value is: 00000 0000 1100 1100
- The complete instruction is: 00001 00101 00000 00000 0000 1100 1100
- The hexadecimal value of this instruction is 0 9 4 0 0 0 C C h

Memory map is then updated with this value.

The next instruction is a type D-add instruction, with the constant field missing:

add R6,R4,R5

The steps followed to obtain the assembly code for this instruction are

- The op-code of the instruction is obtained from the SRC instruction table; it is 01100
- The register codes for the registers R6, R4 and R5 are obtained from the register table; these are 00110, 00100 and 00101 respectively.
- The 12 bit constant field is unused in this instruction, therefore we encode zeros in its place: 0000 0000 0000
- The complete instruction becomes: 01100 00110 00100 00101 0000 0000 0000
- The hexadecimal value of the instruction is 6 1 8 8 5 0 0 0 h

Memory map is then updated with this value.

The instruction **shl R8,R6, 2** is a type C instruction with the rc field missing. The steps taken to obtain the machine code of the instruction are

- The op-code of the shift left instruction 'shl', obtained from the SRC instruction table, is 11100
- The register codes of R8 and R6 are 01000 and 00110 respectively
- Binary code is used for the immediate data 2: 00000 0000 0000 0010
- The complete instruction becomes: 11100 01000 00110 00000 0000 0000 0010
- The hexadecimal equivalent of the instruction is E 2 0 C 0 0 0 2

Memory map is then updated with this value.

The instruction at the memory address 428 is **sub R9, R7, R8**. This is a type D instruction.

We decode it into the machine language, as follows:

- The op-code of the subtract instruction 'sub' is 01110
- The register codes of R9, R7 and R8, obtained from the register table, are 01001, 00111 and 01000 respectively
- The 12 bit immediate data field is not used, zeros are encoded in its place: 0000 0000 0000

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	090000C8 h
416	ld R5, b	094000CC h
420	add R6, R4, R5	61885000 h
424	shl R8, R6, 2	
428	sub R9, R7, R8	
432	st R9, z	

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	090000C8 h
416	ld R5, b	094000CC h
420	add R6, R4, R5	61885000 h
424	shl R8, R6, 2	E20C0002 h
428	sub R9, R7, R8	
432	st R9, z	

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	090000C8 h
416	ld R5, b	094000CC h
420	add R6, R4, R5	61885000 h
424	shl R8, R6, 2	E20C0002 h
428	sub R9, R7, R8	724E8000 h
432	st R9, z	

Advanced Computer Architecture-CS501

4. The complete instruction becomes: 01110 01001 00111 01000 0000 0000 0000
5. The hexadecimal equivalent is 7 2 4 E 8 0 0 0 h

We again update the memory map

The last instruction is is a type C instruction with the rb field missing:

st R9, z

The machine equivalent of this instruction is obtained through the following steps:

1. The op-code of the store instruction 'st', obtained from the SRC instruction table, is 00011
2. The register code of R9 is 01001
3. Notice that there is no register coded in the 5 bit rb field, therefore, we encode zeros: 00000
4. The value of the label z is provided by the assembler, and should be converted to 17 bits. Notice that the memory address assigned to z is 212. The 17 bit binary equivalent is: 00000 0000 1101 0100
5. The complete instruction becomes: 00011 01001 00000 00000 0000 1101 0100
6. The hexadecimal form of this instruction is 1 A 4 0 0 0 D 4 h

The memory map, after the conversion of all the instructions, is

We have shown the memory map as an array of 4 byte cells in the above solution.

However, since the memory of the SRC is arranged in 8 bit cells (i.e. memory is byte aligned), the real representation of the memory map is :

Memory Address	Memory Contents	Hexadecimal Memory Contents
200	unknown	
204	unknown	
208	unknown	
212	unknown	
...
400	ld R1, c	084000D0 h
404	addi R3, R1, 58	68C2003A h
408	shl R7, R3, 4	E1C60004 h
412	ld R4, a	090000C8 h
416	ld R5, b	094000CC h
420	add R6, R4, R5	61885000 h
424	shl R8, R6, 2	E20C0002 h
428	sub R9, R7, R8	724E8000 h
432	st R9, z	1A4000D4 h

Example 3: SRC instruction analysis

Identify the formats of following SRC instructions and specify the values in the fields

Solution:

Memory Address	Memory contents
....
400	08 h
401	40 h
402	00 h
403	D0 h
404	68 h
405	C2 h
406	00 h
407	3A h
408	E1 h
409	C6 h
410	00 h
411	04 h
412	09 h
413	00 h
414	00 h
415	C8 h
416	09 h
417	40 h
....

Instruction	format	ra	rb	rc	c1	c2	c3
neg r1, r2							
add r0,r2,r3							
nop							
ld r2,6							
shl r0,r1,3							

Instruction	format	ra	rb	rc	c1	c2	c3
neg r1, r2	D	r1	-	r2	-	-	-
add r0,r2,r3	D	r0	r2	r3	-	-	-
nop	A	-	-	-	-	-	-
ld r2,6	C	r2	-	-	6	-	-
shl r0,r1,3	C	r0	r1	-	-	-	3

