ASSIGNMENT 12

CPU DESIGN FROM ISA

Group No. : 4
Members:
1.Sayan Ghosh(14CS10061)
2.Ashrujit Ghoshal (14CS10060)

AIM:
INSTRUCTION FORMAT DESIGN FOR CPU

ISA GENERAL PURPOSE REGISTERS

There are 8 general purpose registers r0 to r7. Thus we can use 3 bits to encode them. All registers are 16 bit in size.

MEMORY ADDRESS
The memory address is 16-bits in size. The memory width is also 16 bits.

INSTRUCTION SIZE
The instructions can be either 16-bit or 32-bit in size.

GENERAL ASSEMBLY INSTRUCTION FORMAT

```
          ---------  -----------  -----------
  | opcode  || operand 1 || operand 2 |
   ---------  -----------  -----------
```

ADDRESING MODES FOR OPERAND 2

1. Immediate (with 16-bit operand)
2. Register
3. Base-Indexed addressing (with 16-bit displacement)
4. Base addressing (with 16-bit displacement)
5. (Memory) Indirect (with 16-bit displacement)
6. PC relative (with 16 bit displacement)

INSTRUCTION FORMAT DESIGN:

The instruction set defined in the question consists of 4 broad instructions:
1. Load and Store Instructions

2. ALU Instructions
3. Branch Instructions
4. Subroutine Instructions

The first step of our design includes calculating the total number of instructions/ classes of instructions and decide on the opcode bits.

So, we would require 3 bits for encoding the addressing modes for operand 2

OPCODE BITS

We define a total of 10 major instruction classes as follows :
1. load
2. store
3. ALU(add)
4. ALU(sub)
5. ALU(and)
6. ALU(or)
7. ALU(cmp)
8. ALU(mns)
9. jump
10. jal
11. jr

Thus we need 4 opcode bits to encode these 10 instructions.
Further we make the encoding such that the first opcode bit is used to denote whether the instruction is of ALU type or of NON-ALU Type
Thus 0 indicates the instruction is of ALU type and the rest of the 3 bits are used to indicate if the instruction is and, add or any other type of ALU instruction.
If the instruction is of NON-ALU type, then the first opcode bit is 1.

The encoding of different instructions are shown as follows.

Instruction	**Opcode Bits**
load	1000
store	1001
ALU(add)	0000
ALU(sub)	0001
ALU(and)	0010

ALU(or)	0011
ALU(cmp)	0100
ALU(mns)	0101
jump	1010
jal	1100
jr	1110

ADDRESSING MODES

We keep 3 separate bits as a part of operand 2 to encode the addressing modes of operand 2. Addressing mode is not needed for first operand as the first operand is always in register mode. Based on the addressing mode we can decide whether the next (word)16 bits are a part of the current instruction or not.
Thus if the addressing mode is register type, we can be sure that the next 16 bits form a new instruction and the current instruction size is 16 bits.
Otherwise, the next 16 bits form a part of the current instruction.
As Program Counter(PC) is incremented in the Instruction Fetch phase, we will fetch current instruction and check the addressing mode of operand 2.
If the operand 2 is of type register, then PC is incremented by 2. Otherwise PC is incremented by 4.

The encoding of the Addressing Mode bits based on the addressing mode is as follows:

Addressing Mode	**Encoding**
Immediate	001
Register	000
Base-Indexed addressing	010
Base addressing	011
Indirect	100
PC Relative	101

INSTRUCTION FORMAT FOR DIFFERENT INSTRUCTIONS

Load Instruction

> ld dst, src

dst is in register mode. So we assign 3 bits for operand 1.
Now the addressing mode of the src may be of several addressing modes and thus we need to keep separate 3 bits to specify the addressing modes

> li r5, #100

The instruction encoding will be as follows:
opcode bits : 1000 (4 bits)
r5 (3 bits) : 101 (register is r5, so encoding for 5 is 101)
Since src is #100, the addressing mode is immediate.
addressing mode bits: 001 (3 bits)
The remaining 6 bits of current word are treated as don't cares.

Now, as addressing mode is immediate,
the next word will be a part of current instruction.

```
      ----   ----        ----   ------
     | 1000 || 101 || 001 ||xxxxxx|
      ----     ----   ----   ------
     |      16 bit address        |
      ---------------------------
```

> lr r5, r7

The instruction encoding will be as follows:
opcode bits : 1000 (4 bits)
r5 (3 bits) : 101 (register is r5, so encoding for 5 is 101)
Since src is r7, the addressing mode is register.
addressing mode bits: 000 (3 bits)
The next 3 bits will be :111(r7)
Thus, the operand 2 is of 6 bits here.
The remaining 3 bits of current word are treated as don't cares.

Now, as addressing mode is register, the instruction size will be 16 bits.

```
      ----   ----        ---------  ---
     | 1000 || 101 || 000 111 ||xxx|
      ----     ----   ---------  ---
```

>lx r5, 10(r1, r7)

The instruction encoding will be as follows:
opcode bits : 1000 (4 bits)
r5 (3 bits) : 101 (register is r5, so encoding for 5 is 101)
The addressing mode is base indexed.
addressing mode bits: 010 (3 bits)
The next 3 bits will be :001(r1)

The last 3 bits will be :111(r7)
Thus, the operand 2 is of 6 bits here.

Now, as addressing mode is immediate,
the next word will be a part of current instruction.

```
        ----   ----        -------------
      | 1000 || 101 || 010 001 111 |
        ----      ----   -------------
      |      16 bit address        |
       ----------------------------
```

> ldn r5, @10(r1, r7)

The instruction encoding will be as follows:
opcode bits : 1000 (4 bits)
r5 (3 bits) : 101 (register is r5, so encoding for 5 is 101)
The addressing mode is indirect.
addressing mode bits: 100 (3 bits)
The next 3 bits will be :001(r1)
The last 3 bits will be :111(r7)
Thus, the operand 2 is of 9 bits here.

Now, as addressing mode is immediate,
the next word will be a part of current instruction.

```
        ----   ----        -------------
      | 1000 || 101 || 100 001 111 |
        ----      ----   -------------
      |      16 bit address        |
       ----------------------------
```

-->Store Instructions

>st dst, src
Now, the src is always in register mode.

>stx -5(r2), r3
The instruction encoding will be as follows:
opcode bits : 1001 (4 bits)
r5 (3 bits) : 010 (register is r2, so encoding for 2 is 010)
The addressing mode is based.

addressing mode bits: 011 (3 bits)
The next 3 bits will be :011(r3)
Thus, the operand 2 is of 6 bits here.

>stn @-5(r2), r3
The instruction encoding will be as follows:
opcode bits : 1001 (4 bits)
r5 (3 bits) : 010 (register is r2, so encoding for 2 is 010)
The addressing mode is based.
addressing mode bits: 011 (3 bits)
The next 3 bits will be :011(r3)
Thus, the operand 2 is of 6 bits here.

-->ALU Instructions

The ALU instructions are of the type op r1,r2.
Thus in ALU instructions, the first 4 bits are the opcode bits and the next 3 bits contain the operand 1 and the remaining bits are for addressing modes and operand2
Thus in ALU instructions opeand 2 has 3 bits reserved for addressing modes and remaining 6 bits are for registers based on the addressing mode.

For example:
> addi r1, #43.
The encoding of above instruction is as follows:

```
  ----   ----        ----   ------
| 0000 || 001 || 001 ||xxxxxx|
  ----      ----   ----   ------
|      16 bit address        |
 ----------------------------
```

> subn r4, @-120(r2, r6)
The encoding is as follows:

```
  ----   ----        -------------
| 0000 || 100 || 100 010 110 |
  ----      ----   -------------
|      16 bit address        |
 ----------------------------
```

-->Jump Instructions

In jump instructions, we have the 4 bits of opcode as usual. Other than the opcode bits we will have another 3 bits specifying which jump instruction we are looking at and another 1 bit to say whether we are looking at equals or not equals. We also have 1 flag bit.
The encoding of the jump bits are as follows

Jump Instruction	Encoding(3 bits for type of jump + 1 bit for equals/ not equals)
j	0000
jz	0010
jnz	0011
jc	0100
jnc	0101
jv	0110
jnv	0111
jm	1000
jnm	1001

For example, jnm encoding 1001 means that it this jump is encoded as 100 but as it is of jump when not equals type we have the bit 1 at the end. Thus jnm is encoded as 1001 but jm which is of jump when equals type is encoded as 1000.

So, jm label is encoded as

```
  ----   -----        ---------
| 1010 || 1000 || xxxxxxxx|
  ----     -----   ---------
|     16 bit address     |
 ------------------------
```

→ Subroutine Calls

> jal r5, sub

Thus in jal we have the instruction format as opcode, operand 1 and offset.
The offset is PC relative displacement and thus is considered to be 16 bit word.

Thus in this instruction, we have a 4 bit opcode field followed by a 3 bit operand 1 field.
There are 9 bits of don't cares and the address of the subroutine is stored in the next word.
The encoding is as follows:

```
  ----   ----        ----------
| 1100 || 101 || xxxxxxxxx|
  ----      ----   ----------
|      16 bit address     |
 -------------------------
```

> jr r5

The jr instruction consists of opcode followed by operand 1.

Thus the jr instruction is a 16 bit instruction which has opcode field followed by operand 1. Thus it will have 9 don't cares.

The encoding of the above instruction si as follows:

```
  ----   ----         ----------
| 1110 || 101 || xxxxxxxxx|
  ----      ----   ----------
|      16 bit address     |
 -------------------------
```

<u>Summary :</u>

reg indicates any register from r0 to r7 encoded in 3 bits

Instructions	**opcode (4 bits)**	**operand 1 (3 bits)**	**operand 2 (9 bits)**		
		(always in reg mode)	**mode(3 bit)**	**rb2(3 bit)**	**rx2(3 bit)**
li	1000	reg	001	xxx	xxx
lr	1000	reg	000	reg	xxx
lx	1000	reg	010	reg	reg
ldn	1000	reg	100	reg	reg
Instructions	**opcode (4 bits)**	**operand 1 (9 bits)**			**operand 2(3 bits)**
		mode(3 bit)	**rb1(3 bit)**	**rx1(3 bit)**	**always in reg mode**
stx	1001	010	reg	reg	reg
stn		100	reg	reg	reg

Instructions	opcode (4 bits)	operand 1 (3 bits)	operand 2 (9 bits)		
		(always in reg mode)	mode(3 bit)	rb2(3 bit)	rx2(3 bit)
addi	0000	reg	001	xxx	xxx
addn	0000	reg	100	reg	reg
addr	0000	reg	000	reg	xxx
addx	0000	reg	010	reg	reg
subi	0001	reg	001	xxx	xxx
subn	0001	reg	100	reg	reg
subr	0001	reg	000	reg	xxx
subx	0001	reg	010	reg	reg
andi	0010	reg	001	xxx	xxx
andn	0010	reg	100	reg	reg
andr	0010	reg	000	reg	xxx
andx	0010	reg	010	reg	reg
ori	0011	reg	001	xxx	xxx
orn	0011	reg	100	reg	reg
orr	0011	reg	000	reg	xxx
orx	0011	reg	010	reg	reg
mnsi	0101	reg	001	xxx	xxx
mnsn	0101	reg	100	reg	reg
mnsr	0101	reg	000	reg	xxx
mnsx	0101	reg	010	reg	reg
cmp	0100	reg	xxx	xxx	xxx
	opcode(4 bits)	jump instruction encoding		Don't care	
j	1010	0000		xxxxxxxx	
jz	1010	0010		xxxxxxxx	
jnz	1010	0011		xxxxxxxx	
jc	1010	0100		xxxxxxxx	
jnc	1010	0101		xxxxxxxx	
jb	1010	0110		xxxxxxxx	

jnb	1010	0111		xxxxxxxx	
jm	1010	1000		xxxxxxxx	
jnm	1010	1001		xxxxxxxx	
	Opcode(4 bits)	**link register (3 bits)**	**Don't care**		
jal	1100	reg	xxxxxxxxx		
jr	1110	reg	xxxxxxxxx		

<u>Instruction Decoder</u>

The first 4 bits are read and decoded according to the following table.

<u>the first 4 bits</u>	
1000	load
1001	store
0000	ALU(add)
0001	ALU(sub)
0010	ALU(and)
0011	ALU(or)
0100	ALU(cmp)
0101	ALU(mns)
1010	jump
1100	jal
1100	jr

<u>LOAD:</u>(Opcode=1000)

In all the load instructions,the next 3 bits are taken to be destination in register mode. The following 3 bits specify the mode of the src.

If mode is immediate , last 6 bits are ignored.

If mode is register,the next 3 bits indicate the register and the last 3 bits are ignored.

If mode is base or base indexed the the next 3 bits indicate register rb2 and last 3 bits register rb2

If mode is indirect base addressing the the next 3 bits indicate register rb2 and last 3 bits register rb2

STORE:(opcode:1001)
The first 3 bits indicate the mode of the destination
If mode is base or base indexed the the next 3 bits indicate register rb2 and next 3 bits register rb2
If mode is indirect base addressing the the next 3 bits indicate register rb2 and next 3 bits register rb2

The last 3 bits indicate the src1 register

ALU :

1) **ADD** : the opcode is 0000

   The next 3 bits for the operand 1 are always taken to be in reg mode.

   For the operand 2 the following bits specify the mode :
   If mode is immediate , last 6 bits are ignored.
   If mode is register,the next 3 bits indicate the register and the last 3 bits are ignored.
   If mode is base or base indexed the the next 3 bits indicate register rb2 and last 3 bits register rb2

2) **SUB** : the opcode is 0001

   The next 3 bits for the operand 1 are always taken to be in reg mode.

   For the operand 2 the following bits specify the mode :
   If mode is immediate , last 6 bits are ignored.
   If mode is register,the next 3 bits indicate the register and the last 3 bits are ignored.
   If mode is base or base indexed the the next 3 bits indicate register rb2 and last 3 bits register rb2

3) **AND** : the opcode is 0010

   The next 3 bits for the operand 1 are always taken to be in reg mode.

For the operand 2 the following bits specify the mode :
If mode is immediate , last 6 bits are ignored.
If mode is register,the next 3 bits indicate the register and the last 3 bits are ignored.
If mode is base or base indexed the the next 3 bits indicate register rb2 and last 3 bits register rb2

4) **<u>OR</u>** : the opcode is 0011

The next 3 bits for the operand 1 are always taken to be in reg mode.

For the operand 2 the following bits specify the mode :
If mode is immediate , last 6 bits are ignored.
If mode is register,the next 3 bits indicate the register and the last 3 bits are ignored.
If mode is base or base indexed the the next 3 bits indicate register rb2 and last 3 bits register rb2

5) **<u>MNS</u>** : the opcode is 0101

The next 3 bits for the operand 1 are always taken to be in reg mode.

For the operand 2 the following bits specify the mode :
If mode is immediate , last 6 bits are ignored.
If mode is register,the next 3 bits indicate the register and the last 3 bits are ignored.
If mode is base or base indexed the the next 3 bits indicate register rb2 and last 3 bits register rb2

6) **<u>CMP</u>** : the opcode is 0100

<u>JUMP:</u>(Opcode:1010)
The next 4 bits specify the instruction ie jump to be made.The last 8 bits are ignored.The 4 bits are decoded according to the following rules:

4bits	instruction
0000	j
0010	jz
0011	jnz
0100	jc
0101	jnc
0110	jv
0111	jnv
1000	jm
1011	jnm

<u>JAL</u> : the opcode is 1100

The next 3 bits denote the link register
The next 9 bits are don’t cares

<u>JR</u> : the opcode is 1110

The next 3 bits denote the link register
The next 9 bits are don’t cares