CH 5

Multiplexor

ALU

D-FF (Register)

PC

Program Counter (Register)

Memory

Register file

Incrementor
Why just increment 1?

"Because of Alignment"

"Every instruction is 4 bytes long".
How to Implement ADD Instruction to work?
ADD R1, R2, R3

<table>
<thead>
<tr>
<th>OF</th>
<th>RS</th>
<th>RT</th>
<th>RD</th>
<th>SA</th>
<th>FUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

ADD Rd, Rs Rt
Where the "ADD" control signal for ALU comes? "From the instruction field "5–0" [function field] and OPCode(31–26)"
Can the above H/W perform ADDi Instructions?"

What do we need?
ADD $R7, $R8, 5

<table>
<thead>
<tr>
<th>OP</th>
<th>RS</th>
<th>RT</th>
<th>IMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>8</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

We need to get the immediate data, and "RT" field becomes acting like destination field at "R" format.
Where the "CNT" signal come?
"From the Opcode field (31-26)
(Probably not work due to fan-out)
"Can the above H/W perform OR Instruction?"

"Do we need more H/W?"
OR R9, R10, R11
OR rd, re, rt

<table>
<thead>
<tr>
<th>OP</th>
<th>RS</th>
<th>RT</th>
<th>RD</th>
<th>SA</th>
<th>FUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>11</td>
<td>9</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>

PC

A
Instruction Memory

Reg File

Add R1, R2, Rs
SUB R4, R5, R6
ADDI R7, R8, 5
OR R6, R10, R11

ALUOP

It also can perform “ORi” without having more H/W.”
So far we don’t have any instructions to access memory and we don’t have data memory unit yet.

LW R7, 8(R9) : LW rt, address

<table>
<thead>
<tr>
<th>Op</th>
<th>Rs</th>
<th>rt</th>
<th>imm</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>89</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

How to implement this? And what will be ALU operation?
One way to do this is using the ALU to compute memory address and put the output of ALU (computed memory address; (R9+8)) to the address of memory and put the data from the memory register file (R7).

Can we do this?

Memory output should go back to register file and ALU output should go back to register file.
But this time ALU output is memory address and Dout is data. This is a collision.

How can we solve this? => Multiplexor

Where the CNTMUX come?

=> If "LW" instruction, then the mux should select input from memory.
=> So, from Control Signal.
Can the above H/W perform "LB"?
What will be problems?
How can you modify above H/W to perform "LB"?
LB is loading only one byte and LW is loading 4 bytes. So we have to think about either modifying memory unit (Memory Module) to get one byte data at a time (LW, we assume the memory unit giving us 4 bytes data).

So let's assume the memory unit giving us 4 bytes data, then we have to truncate extra 3 bytes data, and we have to think about memory address.

If we assume we could access arbitrary memory address, then we need logic to do that.

LB R1 2(R3)  LB rt, address

<table>
<thead>
<tr>
<th>OP</th>
<th>Rs</th>
<th>rt</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

31 28 25 21 20 16 15 0
We need logic to truncate extra 3 bytes data and memory address logic to access arbitrary address (Alignment problem).

So when we perform “LBU”, truncation logic will give us byte data with 20 2s rookies padded to upper part. So we could make TRUC_CNT and MEM_CNT from Control Signal. It (LBU) can be done with above H/W. But think about “SB”. Can “SB” be done with above approach?
TRUNCATE LOGIC

MUX-select (from Control Signal)

if LB => select 24 bits zeroes
if LW => select 24 bits from memory

But because of alignment problem and accessing arbitrary address the logic will be complicated.

<table>
<thead>
<tr>
<th>Address</th>
<th>&lt;= data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 2</td>
<td>3</td>
</tr>
<tr>
<td>0 0 0 3</td>
<td>4</td>
</tr>
</tbody>
</table>

The above logic is working when accessing “0000” but what about accessing “0001”?
The problem of SB is we are saving only one byte out of 4 bytes. So we have to think about how to save data to the memory.

For SW, if we assume can save 4 bytes by giving an address, "SW will be easily implemented.

SW \ R10, 11(R12)
Can "SB" be done with above "HW"?
Can we remove the new datapath?
Can we just by pass through ALU?
When we cover Verilog, we implement ALU unit.

It can perform “ADD”, “SUBTRACTION”, “SLT”, ”AND”, “OR”, “ByPassA” and “ByPassB”.

So, we could bypass Drt value through ALU, we can remove the new datapath.

BUT, The Answer is NO.

No., ALU is performing Address calculation (RS + offset).

So, we can not bypass this time.

Then we need the new datapath.
Again, even we can perform SW, doing “SB” is not OK.

For “LB”, we could use “LW” and then truncate some bits (24 bits). But for “SB” case, we can not do that.

One way of doing that is “LW” and modifying byte information and using “SW”.

But this brings extra “LW”.

So, we may assume that we could store byte and for SW instruction, we assume we could store 4 bytes.

Then we need control circuit between Drt and Din.
SB rt. address
SB R10, 11(R12)

<table>
<thead>
<tr>
<th>OP</th>
<th>Rs</th>
<th>rt</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B</td>
<td>12</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

PC

Instruction Memory

A

+1

25-21 20-16

A Read Data

A Read Data

A Write

D/J/N

Address alignment logic

Add MEM

D/J/N

Dout R/W

Sign Extension

Control Signal

MEM data truncation logic

OP FUN for SW, SB, ALUOP

ADD R1, R2, R3
SUB R4, R5, R6
ADDI R7, R8, 5
OR R9, R10, R11
LW R7, 8(R9)
LB R1, 2(R3)
SW R10, 11(R12)
SB R10, 11(R12)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>ADD</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
<td>SUB</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>OR</td>
</tr>
<tr>
<td>23</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>20</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>2b</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>28</td>
<td>X</td>
<td>ADD</td>
</tr>
</tbody>
</table>
Do we need extra datapath to perform Slit?

Do we need extra HW to perform Slit?

Can the above H/W perform "SLTI"?
ox1000: BEQ R10, R11,12

What will be the target address?
How to compute the target address?

BEQ rs, rt, label

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

What kind of action do we need?

a) compute the target address
b) check the branch condition (rs=rt)

Action a) needs Addition
Action b) needs Addition [Subtraction]
Target Address

PC +4 + offset *4

0x1004 [12] 1100
12*4 110000
0x3 0

1004 + 30
ox1034 [ Target address]

---

Diagram a)

PC +4

Offset

Offset*4

Target address

Diagram b)

Rs

Rt

Zero [Rs==Rt]
Target Address Computation logic: Added
Branch condition logic: Added
Multiplexor for PC+4 and Target address: Added

What's wrong with this? What if an instruction performs subtraction and the result is zero; what will be the next address?
Can the previous page H/W perform “BNE”?

a) computing target address

b) computing branch condition
If branch condition is true, the next instruction to be executed is the branch target, instead of PC+4. So, we need to change the H/W.
BNE  R10, R11, 12
BNE  Rs, Rt, label

\[
\begin{array}{cccc}
\text{OP} & \text{rs} & \text{rt} & \text{offset} \\
5 & 10 & 11 & 12 \\
\end{array}
\]

JUMP
\[
\text{J target} \quad j \text{ ox2000}
\]

How to compute the target address?

a) Address computation logic is needed.
Target address computation
{ pc [31:28], target, 00}
10000000 + 2000*4
=10008000 target address

Target address for jump

Target address
Target jump

NPC

PC

A
Instruction Memory
Target address computation
{ pc [31:28], target, 00}

\[10000000 + 2000 \times 4\]
\[= 10008000 \text{ target address}\]

0x10000000  jr 0x2000

0001 0000 0000 0000 0000 0000 0000 0000 (pc)
take 0001 (pc[31:28]) 4bits
take 00 0000 0000 0010 0000 0000 0000 (target) 26bits.

0001 00 0000 0000 0010 0000 0000 0000 00 (padding 2 zeros)
0001 00 0000 0000 0010 0000 0000 0000 00
0001 00 0000 0000 1000 0000 0000 0000 (rearrange)

0X 1 0 0 0 8 0 0 0 (target address)
JUMP
"Can the above H/W perform jr R10?"
Jr R10;
jr Rs

Target address is in R10.
So we need a datapath from D/rs to NPC.

The "Control" comes from
bit 31-26 and bit 5-0
“Jr Rs”
If we want to perform jal, what do we need more?
<table>
<thead>
<tr>
<th>Instruction</th>
<th>OP</th>
<th>FUN</th>
<th>ALUOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>0</td>
<td>20</td>
<td>ADD</td>
</tr>
<tr>
<td>SUB R4, R5, R6</td>
<td>0</td>
<td>22</td>
<td>SUB</td>
</tr>
<tr>
<td>ADDI R7, R8, 5</td>
<td>a</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>OR R9, R10, R11</td>
<td>0</td>
<td>25</td>
<td>OR</td>
</tr>
<tr>
<td>LW R7, 8(R9)</td>
<td>23</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>LB R1, 2(R3)</td>
<td>20</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>SW R10, 11(R12)</td>
<td>2b</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>SB R10, 11(R12)</td>
<td>28</td>
<td>X</td>
<td>ADD</td>
</tr>
<tr>
<td>SLT R1, R2, R3</td>
<td>0</td>
<td>2a</td>
<td>SLT</td>
</tr>
<tr>
<td>SLTI R4, R5, 6</td>
<td>a</td>
<td>X</td>
<td>SLT</td>
</tr>
<tr>
<td>BEQ R10, R11, 12</td>
<td>4</td>
<td>X</td>
<td>SUB</td>
</tr>
<tr>
<td>BNE</td>
<td>5</td>
<td>X</td>
<td>SUB</td>
</tr>
<tr>
<td>J 2000</td>
<td>2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JR R10</td>
<td>0</td>
<td>8</td>
<td>X</td>
</tr>
</tbody>
</table>

For Jal instruction, everything is the same as j instruction, except, we have to save PC+8 or NPC+4 to R31.

Jal 0x2000

\[
\begin{array}{c|c}
3 & 2000 \\
\hline
31 & 29 \\
25 & 0 \\
\end{array}
\]
So we need data path to D/In from NPC and datapath to Awrite to point ra (r31)

This way, we could save PC+8 to ra. The control and CNT signals can be coming from Control logic.
"jal ox2000"
"Added HW for saving R31"
“Jalr Rs, Rd” Instruction.
Everything is same to jr rs instruction, except we have to save the return address to “rd”.

<table>
<thead>
<tr>
<th>Rs</th>
<th>Rf</th>
<th>Rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Jalr  R10, R11

Do we need new datapath?  No
"jalr R10,R11
We don’t need new datapath"