Datapath Elements & Single Cycle Datapath Unit

Chapter 11
Datapath Elements
Introduction

- So far we have discussed many small components and building blocks.

- One final step in our building blocks before we can start to piece together a microprocessor is various datapath elements.
  - We have already discussed portions of these datapath elements in terms of other components and building blocks.
  - We will now consider some of these components and building blocks in ways that will make the design of a microprocessor a little easier in the next chapter.
Register Files
Register Layout

- A general $m \times n$ register file with $m$ registers that are each $n$-bits wide is illustrated below.

- The $r_k$ and $w_i$ signals indicate which register to read/write, respectively.
Register Files
Write Decoder

- For writing to a register, we include a write address with decoder.

- A given *Write Address* (with *Write Enable* = 1) selects which register, 0 through *m* - 1, to store the input from *Data In*. 
Register Files
Write Decoder

- For reading from a register, we include a read address with decoder.

- A given Read Address (with Read Enable = 1) selects which register, 0 through $m - 1$, to read from and output to Data Out.

- Could have multiple data outputs with multiple read address decoders.
Register Files
32-Bit words, 32 Registers

- For the upcoming datapath designs in the next chapter, we want to have a 32x32 register file with one write input and two read outputs.
  - $X_{ra}$ - X read address
  - $Y_{ra}$ - Y read address
  - $Z_{wa}$ - Z write address
  - $X_{do}$ - X data out
  - $Y_{do}$ - Y data out
  - $Z_{di}$ - Z data in
  - rwe - register write enable

- Note: Two data outputs implemented with two read address decoders.
Adder/Subtractor
General Unit Diagram

- An $n$-bit adder/subtractor unit is often illustrated as follows.

- This unit would have $n$ full-adders internally.
Adder/Subtractor
Other Unit Signals

- Other signals often included with an adder/subtractor are shown below.
Logical Unit
Introduction

- A useful unit would be one that can take two $n$-bit inputs and perform some logical operation between each of the bits to get an $n$-bit output.
- For example, given the 8-bit values $0001\,1110$ and $1001\,1000$, we might want to find the bit-wise logical OR.
  \[
  \begin{array}{c}
  \text{bit-wise} \\
  \text{logical OR}
  \end{array}
  \begin{array}{c}
  0001\,1110 \\
  1001\,1000
  \end{array}
  \hline
  1001\,1110
  \]
- Or similarly, the bit-wise logical AND of the two 8-bit values.
  \[
  \begin{array}{c}
  \text{bit-wise} \\
  \text{logical AND}
  \end{array}
  \begin{array}{c}
  0001\,1110 \\
  1001\,1000
  \end{array}
  \hline
  0001\,1000
  \]
- These types of operations are often used for masking and setting bits.
Logical Unit
General Unit Diagram

- Below is a general unit diagram for an \( n \)-bit logical unit.

- Logical operations, such as AND/OR/NOT/NAND/NOR/etc., are done for each bit of \( \textbf{A} \) and \( \textbf{B} \) to form \( \textbf{F} \).
Logical Unit
4-Bit Logical Functions (LF)

- Recall the possible logic functions for two bits, A and B.

- We can use the column $F_n$ as the 4-bit LF input for the logical unit.

| A | B | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Null | Inhibition | $A$ | $B$ | $A + B$ | $\overline{A}$ | $\overline{B}$ | $\overline{A + B}$ | Implication | Identity
Shift Unit
Introduction

- We have already discussed the bulk about shift units in previous chapters.
- As given in the Free-Doc, there are different types of shift units.
  - Logical shift
  - Arithmetic shift
  - Circular shift (this is just a rotate unit)
- We want to discuss an implementation, the barrel shifter, that will be useful in our single cycle datapath computer we will design next chapter.
Shift Unit
General Unit Diagram

- Below is a general unit diagram for an $n$-bit shift unit.

  - $n$-bit value to shift
  - Distance of shift (signed #)
    +’ive = right
    -’ive = left
  - Enable unit (1) or disable unit (0)
  - Shift type
    0 = logical
    1 = arithmetic
    2 = rotate

- Notice that the $n$-bit value $A$ will be shifted according to the distance indicated with signed number $B$. 
Single Cycle Datapath Unit (DPU)

Introduction

- From the previous chapter, we now have a number of datapath elements such as
  - Register file (RF)
  - Adder/subtractor unit (AU)
  - Logical unit (LU)
  - Shift unit (SU)
- The question now is how to take these datapath elements and form a **datapath unit** (DPU).
- The DPU that we will focus on in this chapter is a basic single cycle DPU using a *triple bus internal architecture*. 
Single Cycle DPU
Datapath Elements

- For our examples, we will use the following 32-bit type DPU elements.

Register File (RF)

32x32 x_{do}

Adder/Subtractor Unit (AU)

Logical Unit (LU)

Shift Unit (SU)

- These allow us to design a 32-bit word computer with 32 registers.
- Of course, other word sizes could be used for other designs.
Single Cycle DPU
Add/Subtract Machine

- Below is a simple datapath with a register file and adder/subtractor.

[Diagram showing a 32x32 multiply-accumulate unit with bus connections labeled as Clk, Zwa, Xr, Yr, Zdi, RF, Xbus, Ybus, Datapath, Zbus, and AU/B with a note saying Important: It only takes 1 clock cycle to add/subtract and store the result.]

- This structure is also known as a triple bus internal DPU architecture.
Single Cycle DPU
Add/Subtract Machine

- This simple add/subtract machine DPU allows us to add or subtract values in our registers and store the result back into another register.
- For instance, say that we wanted to add the contents of register R1 with register R2 and store the result back in register R3.

  \[ R3 = R1 + R2 \]

- What control signals are required?
  - \( \bar{a} / s = 0 \) and \( en = 1 \) for AU.
  - \( X_{ra} = 00001, \ Y_{ra} = 00010, \ Z_{wa} = 00011, \) and \( rwe = 1 \) for RF.
- These control signals are applied at the beginning of a clock cycle. The signals then propagate forming the sum at the output of the AU. At the end of the clock cycle, the sum (on Z bus) is clocked into R3.
Single Cycle DPU
Add/Subtract Machine

- What if instead we wanted to perform the following operation.
  \[ R2 = R1 + R2 \]

- The control signals required are
  - \( \overline{a}/s = 0 \) and \( en = 1 \) for AU.
  - \( X_{ra} = 00001, Y_{ra} = 00010, Z_{wa} = 00010, \) and \( rwe = 1 \) for RF.

- What is the result of this if the current value of \( R1=0x00000001 \) and \( R2=0x00000003 \)?
  - The register \( R2 \) would be updated at the end of the clock cycle with the value \( 0x00000004 \).

- Remember, the current value of \( R2 \) is put on the X or Y bus, and it is only at the END of the clock cycle that the contents of \( R2 \) get changed.
Single Cycle DPU
Basic Single Cycle DPU

- A more useful single cycle datapath can be as follows.

- This structure is still a triple bus internal DPU architecture.
Single Cycle DPU
Computation Examples

• How does this change the additions we were doing/earlier?
  • Say we want to again perform the following addition.

\[ R3 = R1 + R2 \]

• The control signals we would need are
  • \( \bar{a}/s = 0 \) and \( en = 1 \) for AU.
  • \( en = 0 \) for LU.
  • \( en = 0 \) for SU.
  • \( X_{ra} = 00001, Y_{ra} = 00010, Z_{wa} = 00011, \) and \( rwe = 1 \) for RF.

• Notice that we use the same control signals as before, but now include signals to disable the LU and SU during this addition clock cycle.
Single Cycle DPU
Computation Examples

- Another operation we might want to do with this DPU is perform a logical shift of the contents of **R15** by a distance indicated in **R6**.
  - The control signals required are
    - \( en = 0 \) for **AU**.
    - \( en = 0 \) for **LU**.
    - \( en = 1 \) and \( ST = 00 \) for **SU**.
    - \( X_{ra} = 01111 \), \( Y_{ra} = 00110 \), \( Z_{wa} = 01111 \), and \( rwe = 1 \) for **RF**.
  - Notice that this set of control signals disables the **AU** and **LU** while enabling the **SU**.
  - The **SU** is set to do a logical shift with \( ST = 00 \).
  - The distance of the shift is according to what is in **R6**.
  - The result is stored back in **R15** with \( Z_{wa} = 01111 \) and \( rwe = 1 \) for **RF**.
Single Cycle DPU
Arithmetic Logic Unit

- Since only one of AU, SU, or LU will be active at a time in this architecture, we will combine to form an **arithmetic logic unit** (ALU).
Single Cycle DPU
Single Cycle DPU With ALU

• Using our ALU, the DPU can be redrawn as follows.

• This structure is still a triple bus internal DPU architecture.
Single Cycle DPU
Immediate Register

- Many designs also include some form of immediate register.

- Allows for operations such as $R28 = R5 + \text{Immediate}$. 
Single Cycle DPU
Immediate Register

- The `im_en` line does two things:
  - When 0, `im_en` controls
    - immediate register outputs to go to high impedance so as NOT to affect Y bus.
    - register file Y data out to output corresponding register value.
  - When 1, `im_en` controls
    - immediate register to output register value to Y bus.
    - register file Y data out to go to high impedance so as NOT to affect Y bus.
- The `im_va` lines pass a value to the immediate register.
32x32 bits is not sufficient memory for most computers.

We can include external memory (SRAM, DRAM, etc.) as follows.
Single Cycle DPU
Including Memory

- The included has the following characteristics
  - 32 address lines
  - 32 data lines
  - a read/write line
  - a chip select or memory select line
- Two transmission gates block the bidirectional data lines for the memory.
  - Notice that if `st_en` is high, then we can potentially write to the memory.
  - Notice that if `ld_en` is high, then we can potentially read from the memory.
Single Cycle DPU
Reading from Memory

- We wish to be able to read and write from our memory.
- A sample read/load operation can be expressed as follows

\[ R4 = M[R7] \]

- This operation uses the value in \( R7 \) as the address to the memory and reads the value at that address in the memory to \( R4 \).
- What control signals are required?
  - \( \text{en} = 0 \) for ALU.
  - \( X_{ra} = 00111, Y_{ra} = XXXXX, Z_{wa} = 00100, \) and \( r\text{we} = 1 \) for RF.
  - \( \text{st}_{-}\text{en} = 0 \) and \( \text{id}_{-}\text{en} = 1 \)
  - \( \sim r/w = r \) and \( m\text{sel} = 1 \)
Single Cycle DPU
Writing to Memory

- A sample write/store operation can be expressed as follows

\[ M[R5] = R9 \]

- This operation uses the value in \( R5 \) as the address to the memory and write the value in \( R9 \) to that address in the memory.
- What control signals are required?
  - \( \text{en} = 0 \) for ALU.
  - \( X_{ra} = 00101, Y_{ra} = 01001, Z_{wa} = XXXXX, \) and \( rwe = 0 \) for RF.
  - \( st\_en = 1 \) and \( ld\_en = 0 \)
  - \( \sim r/w = w \) and \( msel = 1 \)
Single Cycle DPU
Microcode

- Microcode in a processor are all of the control signals required to execute an operation for a clock cycle.
- We have actually looked at examples of a microcode operation when we considered various operations such as
  \[ R3 = R1 + R2 \]
  or
  \[ M[R5] = R9 \]
Single Cycle DPU
Controller Signals

- In general, these control signals can be burned into a ROM.

- Each opcode has its own set of general control signals for the DPU.
Single Cycle DPU
Controller Signals

- For our DPU, the control signals are as follows.

```
  opcode
    6

  rwe
  imm en
  au en
  al/s
  lu en
  if (4 bits)
  su en
  st (2 bits)
  st en
  ld en
  riw
  msel
```

ROM (OPCODES)
Single Cycle DPU
DPU with Controller

- Now, an input opcode will send appropriate control signals to the DPU for that major operation.

- Notice that we still need register addresses and the immediate value.
Instructions
Instruction Format

- While instructions can come in many different shapes and forms, we will consider the following 32-bit instruction formats to loosely follow the MIPS R3000/4000 format.

<table>
<thead>
<tr>
<th>R-format</th>
<th>I-format</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 25 20 15 10 0</td>
<td>31 25 20 15 0</td>
</tr>
<tr>
<td>opcode Z X Y other potential bits</td>
<td>opcode Z X immediate value</td>
</tr>
</tbody>
</table>
Instructions
R-Format with DPU

- If we have an R-format instruction, we link the bits as follows.

![Diagram of R-Format with DPU]
Instructions
I-Format with DPU

- If we have an I-format instruction, we link the bits as follows:

```
31 25 20 15 0
opcode Z X immediate
```

Notice sign extension of 16-bit value.
Instructions
Instruction Register

- Use a general instruction register that can act as R- or I-Format.