Lecture 6

Bus Architecture and I/O Interfacing

Asst. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering
İzmir Institute of Technology
PCI Express bus card slots (from top to bottom: x4, x16, x1 and x16), compared to a traditional 32-bit PCI bus card slot (bottom)
Instructions

(1) mov reg, reg
(2) movx @A, reg
(3) movx reg, @A
(4) movi reg, imm8
(5) adda reg
(6) suba reg
(7) inc reg
(8) dec reg
(9) anda reg
(10) ora reg
(11) not reg
(12) jnz add8
(13) call add8
(14) ret
(15) jmp add8
(16) nop
Instruction Formats

All instructions are fixed to 8-bit long except for `call, jmp, jnz and movi`:

For `jnz, jmp and call`:

| opcode | unused | address |

For `ret`:

| opcode | unused |

For `mov`:

| opcode | dest | src |

For `movi`:

| opcode | dest | immediate |
ALU and Register File

- Asel1-0
- ALUSel4-0
- IR
- ALU
- WA1-0
- RBA1-0
- Acc
- 4x8 RegFile
- IR3-2
- IR1-0
- we
- rbe

Izmir Institute of Technology
Real-Time and Embedded System Design
PC Next Logic and Stack File

"000" & databus_{2-0} & "11"

Ssel \rightarrow 012

PC

PCload
Reset

sub

jmpMux

0 1

IR_{(imm)}

sub := jmpMux and IR_7

To the Address Input of Program Memory

STACK

push

pop
entity stack is port(
  clk: in std_logic;
  reset: in std_logic;
  push: in std_logic;
  pop: in std_logic;
  input: in std_logic_vector(7 downto 0);
  output: out std_logic_vector(7 downto 0));
end stack;
architecture imp of stack is
  subtype reg is std_logic_vector(7 downto 0);
  type regArray is array(0 to 3) of reg;
  signal SF: regArray;
begin
  PUSHPOP: Process(clk)
  variable index: integer:=3;
  begin
    if(clk'event and clk='1') then
      if(reset='1') then
        SF(0) <= X"00"
        SF(1) <= X"00"
        SF(2) <= X"00"
        SF(3) <= X"00"
        index := 3;
      elsif(push='1') then
        SF(index) <= input+1;
        index := index - 1;
        if(index = -1) then
          index := 7;
        end if;
      elsif(pop='1') then
        index := index + 1;
        if(index = 8) then
          index := 0;
        end if;
        output <= SF(index);
      end if;
    end if;
  end process:
end architecture;
Making Address and Data Busses

ALUSel4-0

4x8 RegFile

we

WA1-0

we

IR3-2

rbe

B

RBA1-0

IR1-0

Acc

8-bit address bus

8-bit data bus

?
PC, IR and Program Memory

PC

IR

IR (Imm. and Add.)

IR4-0 (To Register File)

IR7-5 (To Control Unit)

A7-0  D7-0

256x8 ROM

CS

opfetch
Modification to Control Unit

clk  →  ALUSel
reset  →  Asel
Aeq  →  writeAcc
IR7-5  →  IRLoad
  →  PClload
  →  Oload
  →  jmpMux
  →  opfetch
WR  →  we
RD  →  rbe
INT  →
INTA  →
Example 1

Connecting two 8255s to provide 24-bit input and 24-bit output
uPabs-2 with 48-bit port
Programming 8255 IC

Address “00” -> Port A
Address “01” -> Port B
Address “02” -> Port C
Address “03” -> Command word
Address Decoder Circuits for 8255s

• Partial Address Decoding:
  – We can use one bit A(7) signal for the first address decoder and A(7)' (using inverter) for the second address decoder.

• Full Address Decoding:
  – Place the 8255s as seen in the memory map:
Programming

• Initializing 8255s:

```
MOVI A, 03H
MOVI B, 80H
MOVX @A, B

MOVI A, 07H
MOVI B, 9BH
MOVX @A, B
```

• Writing “0”s to the output ports:

```
MOVI B, 00H
MOVI A, 00H
MOVX @A, B
MOVI A, 01H
MOVX @A, B
MOVI A, 02H
MOVX @A, B
```
Example 2

- Requirements:
  - The clock period of uPabs is 1μs.
  - Short commands take 4 clock cycles. Long commands take 8 clock cycles.
  - uPabs is supposed to generate a square wave signal at the port pin PA(0) of the first 8255. The period will be approx. 100 μs.
  - uPabs should also read one bit (from one of PA pins of the second 8255). If this bit is “0” uPabs should do nothing. If it is “0”, then uPabs must immediately set the PA(0) to “1”. PA(0) stays there even if the input returns to “0”.

- Design the circuit.
- Write an assembly program.
Assembly Program for Example 2

**Initialize:**

- `MOVI A, 03H`
- `MOVI B, 80H`
- `MOVX @A, B`
- `MOVI A, 07H`
- `MOVI B, 9BH`
- `MOVX @A, B`
- `MOVI B, 00H`
- `MOVI A, 00H`
- `MOVX @A, B`
- `MOVI A, 01H`
- `MOVX @A, B`
- `MOVI A, 02H`
- `MOVX @A, B`

**Square wave:**

- `MOV A, D`
- `MOV A, 00H`
- `MOVX @A, C`
- `NOT C`
- `MOV A, 00H`
- `MOVX @A, C`

**Stopping condition:**

- `DEC D`
- `JNZ LOOP`
- `NOT C`
- `MOV A, 00H`
- `MOVX @A, C`
- `JMP LOOP`

*Is timing (the period of square wave) correct?*
Questions

• Find ??H in MOV D, ??H command that defines the period of the square wave
• Embed the stopping code into the square wave generator program provided that the timing satisfies the requirements.
• Discuss about the timing errors and jitters.
• Convert your assembly code to the machine code.