Lecture 18
PicoBlaze Instruction Set & Assembler Directives
Required reading

- P. Chu, *FPGA Prototyping by VHDL Examples*  
  *Chapter 15 PicoBlaze Assembly Code Development*

Recommended reading

- K. Chapman, *PicoBlaze for Spartan-6, Virtex-6, and 7-Series (KCPSM6)*
PicoBlaze-6 Programming Model

Bank A
- s0
- s1
- s2
- s3
- sc
- sd
- se
- ef

Bank B

register file
data RAM
Instruction memory
call/return stack
flags

pc

tos

00
01
02
03

000
001
002
003

FFC
FFD
FFE
FFF

c
z
i
## Syntax and Terminology

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sX</td>
<td>s7</td>
<td>Value at register 7</td>
</tr>
<tr>
<td>KK</td>
<td>0xAB</td>
<td>Constant AB (in hex)</td>
</tr>
<tr>
<td>PORT(KK)</td>
<td>PORT(2)</td>
<td>Input value from port 2</td>
</tr>
<tr>
<td>PORT((sX))</td>
<td>PORT((sa))</td>
<td>Input value from the port specified by register a</td>
</tr>
<tr>
<td>RAM(KK)</td>
<td>RAM(4)</td>
<td>Value from RAM location 4</td>
</tr>
</tbody>
</table>
Addressing modes

Immediate mode

SUB \( s7, 0x07 \)
ADDCC \( s2, 0x08 \)

Direct mode

ADD \( sa, sf \)
IN \( s5, 0x2a \)

Indirect mode

STORE \( s3, (sa) \)
IN \( s9, (s2) \)

\( s7 \leq s7 - 0x07 \)
\( s2 \leq s2 + 0x08 + C \)

\( sa \leq sa + sf \)
\( s5 \leq \text{PORT}(0x2a) \)

\( \text{RAM}((sa)) \leq s3 \)
\( s9 \leq \text{PORT}((s2)) \)
# PicoBlaze Development Environments

<table>
<thead>
<tr>
<th>Feature</th>
<th>Xilinx KCPSM3/6</th>
<th>FIDEx IDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform support</td>
<td>Windows</td>
<td>Windows XP, Windows 7 and 8</td>
</tr>
<tr>
<td>Assembler</td>
<td>Command-line in DOS window</td>
<td>Graphical</td>
</tr>
<tr>
<td>Instruction Syntax</td>
<td>KCPSM3/6</td>
<td>FIDEx IDE</td>
</tr>
<tr>
<td>Instruction Set Simulator</td>
<td>Facilities provided for VHDL simulation</td>
<td>Graphical/ Interactive</td>
</tr>
<tr>
<td>Simulator Breakpoints</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>Register Viewer</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>Memory Viewer</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>
KCPSM6 Assembler Files

Block RAM initialization templates for a variety of design flows:
- ROM_form.vhd
- ROM_form.v
- ROM_form.coe

Assembled PicoBlaze code, formatted to initialize a block RAM for a variety of design flows:
- <filename>.vhd
- <filename>.v
- <filename>.coe
- <filename>.m

Assembled PicoBlaze code, formatted for other utilities:
- <filename>.hex
- <filename>.dec

PicoBlaze source program:
- <filename>.psm

Assembler intermediate processing files (possibly useful for debugging assembly errors):
- pass1.dat
- pass2.dat
- pass3.dat
- pass4.dat
- pass5.dat

Assembler report files:
- <filename>.log
- constants.txt
- labels.txt

Formatted version of input source program:
- <filename>.fmt
# Directives of Assembly Language

<table>
<thead>
<tr>
<th>Function</th>
<th>KCPSM3/6 Directive</th>
<th>FIDEx IDE Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locating Code</td>
<td>ADDRESS 3FF</td>
<td>ORG 0x3FF</td>
</tr>
<tr>
<td>Aliasing Register Names</td>
<td>NAMEREG s5, myregname</td>
<td>myregname EQU s5</td>
</tr>
<tr>
<td>Declaring Constants</td>
<td>CONSTANT myconstant, 80</td>
<td>myconstant EQU 0x80</td>
</tr>
<tr>
<td>Naming the program ROM file</td>
<td>Name using the same base filename as the assembler source file</td>
<td>Specify the VHDL template file in the project settings under processor tab and compiler options</td>
</tr>
<tr>
<td>Equating symbolic name for an I/O port ID</td>
<td>N/A</td>
<td>sw_port EQU 0x01 ; 8-bit switches sw_in EQU sf IN sw_in, sw_port ; read switch input data EQU s0 ; reg for tmp data led_port EQU 0x05 OUT data, led_port</td>
</tr>
</tbody>
</table>
Differences between Mnemonics of Instructions

<table>
<thead>
<tr>
<th>KCPSM3/ 6 Mnemonic</th>
<th>FIDEx IDE Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN</td>
<td>RET</td>
</tr>
<tr>
<td>RETURN C</td>
<td>RET C</td>
</tr>
<tr>
<td>RETURN NC</td>
<td>RET NC</td>
</tr>
<tr>
<td>RETURN Z</td>
<td>RET Z</td>
</tr>
<tr>
<td>RETURN NZ</td>
<td>RET NZ</td>
</tr>
<tr>
<td>RETURNI ENABLE</td>
<td>RETI ENABLE</td>
</tr>
<tr>
<td>RETURNI DISABLE</td>
<td>RETI DISABLE</td>
</tr>
<tr>
<td>ENABLE INTERRUPT</td>
<td>EINT</td>
</tr>
<tr>
<td>DISABLE INTERRUPT</td>
<td>DINT</td>
</tr>
</tbody>
</table>
# Differences between Mnemonics of Instructions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD CY sX, sY</td>
<td>ADDC sX, sY</td>
</tr>
<tr>
<td>SUB CY sX, sY</td>
<td>SUBC sX, sY</td>
</tr>
<tr>
<td>INPUT sX, (sY)</td>
<td>IN sX, (sY)</td>
</tr>
<tr>
<td>INPUT sX, kk</td>
<td>IN sX, kk</td>
</tr>
<tr>
<td>OUTPUT sX, (sY)</td>
<td>OUT sX, (sY)</td>
</tr>
<tr>
<td>OUTPUT sX, (sY)</td>
<td>OUT sX, (sY)</td>
</tr>
<tr>
<td>COMPARE sX, sY</td>
<td>COMP sX, sY</td>
</tr>
</tbody>
</table>
Differences between Programs

```
CONSTANT myconstant

NAMEREG a0, count16_lsb
NAMEREG a1, count16_msb

ADDRESS 000

main:
    ; initialize 16-bit counter, enable interrupts
    LOAD count16_lsb, myconstant
    ENABLE INTERRUPT

loop:
    ; continuously increment 16-bit count
    CALL increment_count
    JUMP loop
end_main:

increment_count:
    ; add 1 to LSB of 16-bit counter
    ADD count16_lsb, 01
    ; only add one to MSB if carry generated by LSB
    ADDCY count16_msb, 00
    RETURN

isr:
    ; decrement 16-bit counter by one on interrupt
    ; subtract 1 from LSB of 16-bit counter
    SUB count16_lsb, 01
    ; only subtract one from MSB if borrow generated by LSB
    SUBCY count16_msb, 00
    RETURN ENABLE

; interrupt vector is always in last memory location
ADDRESS 3FF
; jump to interrupt service routine (ISR)
JUMP isr
```

```
myconstant EQU 0xA5

count16_lsb EQU a0
count16_msb EQU a1

ORG 0x000

main:
    ; initialize 16-bit counter, enable interrupts
    LOAD count16_lsb, myconstant
    ENABLE INTERRUPT

loop:
    ; continuously increment 16-bit count
    CALL increment_count
    JUMP loop
end_main:

increment_count:
    ; add 1 to LSB of 16-bit counter
    ADD count16_lsb, 0x01
    ; only add one to MSB if carry generated by LSB
    ADDCY count16_msb, 0x00
    RET

isr:
    ; decrement 16-bit counter by one on interrupt
    ; subtract 1 from LSB of 16-bit counter
    SUB count16_lsb, 0x01
    ; only subtract one from MSB if borrow generated by LSB
    SUBCY count16_msb, 0x00
    RETI ENABLE

; interrupt vector is always in last memory location
ORG 0x3FF
; jump to interrupt service routine (ISR)
JUMP isr
```
Basic Data Movement

Instructions
Data Movement Instructions

LOAD

LOAD \( sX, sY \)
\( sX \leq sY \)

LOAD \( sX, KK \)
\( sX \leq KK \)
Logic Instructions
& Bit Manipulation
Logic instructions

1. **AND**
   AND sX, sY
   sX <= sX and sY
   AND sX, KK
   sX <= sX and KK

2. **OR**
   OR sX, sY
   sX <= sX or sY
   OR sX, KK
   sX <= sX or KK

3. **XOR**
   XOR sX, sY
   sX <= sX xor sY
   XOR sX, KK
   sX <= sX xor KK

   \[\begin{array}{|c|c|c|}
   \hline
   & IMM, DIR & \text{C Z} \\
   \hline
   0 & \uparrow & 0 \\
   \hline
   \end{array}\]
Questions

Perform the following operations in the assembly language of PicoBlaze

1. Set the most significant bit of the register s0 to 1

2. Clear two middle bits of the register s1

3. Toggle the least significant bit of the register s2
Answers

1. Set the most significant bit of the register $s0$ to 1
   
   BIT7 EQU 0x80
   OR $s0$, BIT7

2. Clear two middle bits of the register $s1$
   
   BITS43C EQU 0xE7
   AND $s1$, BITS43C

3. Toggle the least significant bit of the register $s2$
   
   BIT0 EQU 0x01
   XOR $s2$, BIT0
Arithmetic Instructions

& Multiple-Byte Manipulation
Arithmetic Instructions (1)

Addition

**ADD sX, sY**

\[ sX \leq sX + sY \]

**ADD sX, KK**

\[ sX \leq sX + KK \]

**ADDC sX, sY**

\[ sX \leq sX + sY + \text{CARRY} \]

**ADDC sX, KK**

\[ sX \leq sX + KK + \text{CARRY} \]

<table>
<thead>
<tr>
<th>IMM, DIR</th>
<th>C Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>▲ ▲</td>
</tr>
</tbody>
</table>
Questions
Perform the following operation in the assembly language of PicoBlaze

Add two 3-byte numbers $X=(x_2, x_1, x_0)$ and $Y=(y_2, y_1, y_0)$, stored originally in registers $(s_2, s_1, s_0)$ and $(s_5, s_4, s_3)$, accordingly.
Answer

\[\text{ADD } x_0, y_0\]
\[\text{ADDC } x_1, y_1\]
\[\text{ADDC } x_2, y_2\]
Arithmetic Instructions (2)

Subtraction

SUB sX, sY
sX <= sX – sY

SUB sX, KK
sX <= sX – KK

SUBC sX, sY
sX <= sX – sY – CARRY

SUBC sX, KK
sX <= sX – KK – CARRY
Questions

Perform the following operation in the assembly language of PicoBlaze

Perform the subtraction $X = X - Y$, where $X = (x_2, x_1, x_0)$ and $Y = (y_2, y_1, y_0)$, and these variables are originally stored in registers $(s_2, s_1, s_0)$ and $(s_5, s_4, s_3)$, accordingly.
Answer

x0 EQU s0
x1 EQU s1
x2 EQU s2
y0 EQU s3
y1 EQU s4
y2 EQU s5

SUB x0, y0
SUBC x1, y1
SUBC x2, y2
Shifts & Rotations
### Edit instructions - Shifts

<table>
<thead>
<tr>
<th>Shift Left</th>
<th>Shift Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SL0</strong></td>
<td><strong>SR0</strong></td>
</tr>
<tr>
<td>Shift Left with '0' fill.</td>
<td>Shift Right with '0' fill.</td>
</tr>
<tr>
<td>CARRY</td>
<td>Register ax</td>
</tr>
<tr>
<td></td>
<td>‘0’ ← 7 6 5 4 3 2 1 0 → CARRY</td>
</tr>
<tr>
<td><strong>SL1</strong></td>
<td><strong>SR1</strong></td>
</tr>
<tr>
<td>Shift Left with '1' fill.</td>
<td>Shift Right with '1' fill.</td>
</tr>
<tr>
<td>CARRY</td>
<td>Register ax</td>
</tr>
<tr>
<td></td>
<td>‘1’ ← 7 6 5 4 3 2 1 0 → CARRY</td>
</tr>
<tr>
<td><strong>SLX</strong></td>
<td><strong>SRX</strong></td>
</tr>
<tr>
<td>Shift Left, eXtend bit 0.</td>
<td>Shift Right, sign eXtend.</td>
</tr>
<tr>
<td>CARRY</td>
<td>Register ax</td>
</tr>
<tr>
<td></td>
<td>7 6 5 4 3 2 1 0 → CARRY</td>
</tr>
<tr>
<td><strong>SLA</strong></td>
<td><strong>SRA</strong></td>
</tr>
<tr>
<td>Shift Left through All bits, including CARRY.</td>
<td>Shift Right through All bits, including CARRY.</td>
</tr>
<tr>
<td>CARRY</td>
<td>Register ax</td>
</tr>
<tr>
<td></td>
<td>7 6 5 4 3 2 1 0 → CARRY</td>
</tr>
</tbody>
</table>

*All shift instructions affect Zero and Carry flags*
Edit instructions - Rotations

*All rotate instructions affect Zero and Carry flags*
Questions
Perform the following operation in the assembly language of PicoBlaze

Perform the left shift \( (C, X) \leq X \ll 1 \),
where \( X = (x_2, x_1, x_0) \) is stored in registers \((s_2, s_1, s_0)\),
and the most significant bit of \( X \) is shifted into the carry flag.
Answer

\[
\begin{align*}
  x_0 & \quad \text{EQU} \quad s_0 \\
  x_1 & \quad \text{EQU} \quad s_1 \\
  x_2 & \quad \text{EQU} \quad s_2 \\
  \text{SL0} & \quad x_0 \\
  \text{SLA} & \quad x_1 \\
  \text{SLA} & \quad x_2
\end{align*}
\]
Test, Compare, and Program Flow Instructions
Test and Compare Instructions

**TEST**

TEST \( sX, sY \)
\( sX \) and \( sY \) \( \Rightarrow \) none

TEST \( sX, KK \)
\( sX \) and \( KK \) \( \Rightarrow \) none

**COMPARE**

COMP \( sX, sY \)
\( sX - sY \) \( \Rightarrow \) none

COMP \( sX, KK \)
\( sX - KK \) \( \Rightarrow \) none

C Z
\( \uparrow \uparrow \uparrow \)

\( C = \) odd parity of the result

<table>
<thead>
<tr>
<th>Flag</th>
<th>When Flag=0</th>
<th>When Flag=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>( \text{Operand}_1 \neq \text{Operand}_2 )</td>
<td>( \text{Operand}_1 = \text{Operand}_2 )</td>
</tr>
<tr>
<td>CARRY</td>
<td>( \text{Operand}_1 &gt; \text{Operand}_2 )</td>
<td>( \text{Operand}_1 &lt; \text{Operand}_2 )</td>
</tr>
</tbody>
</table>
Program Flow Control Instructions (1)

**JUMP** AAA

PC <= AAA

**JUMP** C, AAA

if C=1 then PC <= AAA else PC <= PC + 1

**JUMP** NC, AAA

if C=0 then PC <= AAA else PC <= PC + 1

**JUMP** Z, AAA

if Z=1 then PC <= AAA else PC <= PC + 1

**JUMP** NZ, AAA

if Z=0 then PC <= AAA else PC <= PC + 1
If-else statement

C

if (s0 == s1) {
    ........
}
else {
    ........
}
**If-else statement**

### C

```c
if (s0 == s1) {
    ........
}
else {
    ........
}
```

### Assembly language

```assembly
COMP s0, s1
JUMP NZ, else_branch
    ........
JUMP if_done
else_branch:
    ........
if_done:
```
Switch statement

C

switch (s0) {
  case value_1:
      ........
      break;
  case value_2:
      ........
      break;
  case value_3:
      ........
      break;
  default:
      ........
      break;
}

Assembly language
Switch statement

C

switch (s0) {
    case value_1:
        ........
        break;
    case value_2:
        ........
        break;
    case value_3:
        ........
        break;
    default:
        ........
        break;
}

Assembly language

    COMP s0, value_1
    JUMP NZ, case_2
    ........
    JUMP case_done

case_2:
    COMP s0, value_2
    JUMP NZ, case_3
    ........
    JUMP case_done

case_3:
    COMP s0, value_3
    JUMP NZ, default
    ........
    JUMP case_done

default:
    ........

case_done:
C

for(i=MAX, i>=0, i--) {
   .............
}

Assembly language
C

for(i=MAX; i>0; i--) {
    ............
}

for(i=MAX; i>=0; i--) {
    ............
}
Subroutines
Subroutine-Related Instructions

CALL  AAA
    TOS <= TOS+1; STACK[TOS] <= PC; PC <= AAA

CALL  C | Z , AAA
    if C | Z =1  then
        TOS <= TOS+1; STACK[TOS] <= PC; PC <= AAA
    else
        PC <= PC + 1

CALL  NC | NZ , AAA
    if C | Z =0  then
        TOS <= TOS+1; STACK[TOS] <= PC; PC <= AAA
    else
        PC <= PC + 1
Subroutine-Related Instructions

RET

PC <= STACK[TOS] + 1; TOS <= TOS - 1

RET C | Z
if C | Z =1 then
   PC <= STACK[TOS] + 1; TOS <= TOS - 1
else
   PC <= PC + 1

RET NC | NZ
if C | Z =0 then
   PC <= STACK[TOS] + 1; TOS <= TOS - 1
else
   PC <= PC + 1
Subroutine Call Flow

; ====== main program ======

...  

1

add s0, s3

call routine1

sub s5, 01

...  

5

; ====== routine1 ======

routine1:

...  

3

return

...  

2
Instructions Involving Data Memory
Data Movement Instructions
Involving Data Memory

STORE

STORE  sX, KK
RAM(KK) <= sX

STORE  sX, sY
RAM(sY) <= sX

FETCH

FETCH  sX, KK
sX <= RAM(KK)

FETCH  sX, (sY)
sX <= RAM(sY)
Example 1: Clear Data RAM of the size of 64 bytes

;---------------------------------------------------------------
; routine: clr_data_mem
; function: clear data ram
; temp register: data, s2
;---------------------------------------------------------------

RAM_SIZE   EQU   0x40    ; size of RAM = 64
clr_data_mem:    
    load s2, RAM_SIZE    ; unitize loop index to 64
    load s0, 0x00

clr_mem_loop:    
    sub s2, 0x01          ; dec loop index
    store s0, (s2)
    jump nz, clr_mem_loop ; repeat until s2=0
ret
Input/Output

Instructions
Input/Output Instructions

IN

IN sX, KK
sX <= PORT(KK)
IN sX, (sY)
sX <= PORT((sY))

OUT

OUT sX, KK
PORT(KK) <= sX
OUT sX, (sY)
PORT((sY)) <= sX

DIR, IND

C Z
---
Example 2: Clear External RAM of the size of 64 bytes

; routine: clr_ext_mem
; function: clear data ram
; temp register: data, s2

RAM_SIZE   EQU   0x40      ; size of RAM = 64
clr_ext_mem:
    load s2, RAM_SIZE       ; unitize loop index to 64
    load s0, 0x00
clr_mem_loop:
    sub s2, 0x01            ; dec loop index
    out s0, (s2)
    jump nz, clr_mem_loop   ; repeat until s2=0
return