Assembly Language Instructions

Milestone 2

General Purpose (8 registers):

- Euclid's Algorithm architecture has 7 visible 16bit general purpose registers ($g1-$g7)
- 16-bit $ra for Return Address

We have the follow special purpose registers called:

- 16-bit PC for the program counter
- 16-bit IR for the Instruction Register
- 16-bit IPC that holds the PC before jumping to an interrupt
- 1- bit EI for Enable Interrupt

We have a special purpose register file containing 8 16-bit interrupt address registers in order to use special instructions for interrupt handling:

- IAR0 thru IAR7

In addition to the interrupt address registers, there will be a 8-bit special purpose register called IntStatus that will hold an interrupt mask that will be set when an interrupt should fire. Each bit in the register corresponds to one of the interrupts. When one of the bits is set to 1, then the PC will be changed to the corresponding interrupt address so that the interrupt instructions will be executed.

We also have two 4-bit input ports, and 16-bit output port. These are known as IN0 and IN1 for the input ports and OUT0 for the output port.

Machine Language Instruction formats

Name	**Fields**					**Comments**
Field size	5 bits	3 bits	3 bits	3 bits	2 bits	All instructions 16 bits
R-Type	Op	Rd	Rs	Rt	unused	Register instruction format
I-Type	Op	Rd	immediate			Immediate instruction type
J-Type	Op	Address				Jump instruction type

Arithmetic Instructions

Addition:

ADD rd, rs

0	Rd	Rs	0	0
5	3	3	3	2

Takes the contents of rd and rs, adds them and stores the answer in rd

ADDI rd, imm

22	Rd	Immediate
5	3	8

Takes Rd and adds the sign extended immediate value. The result is stored in Rd.

Subtraction:

SUB rd, rs

1	Rd	Rs	0	0
5	3	3	3	2

Takes the contents of rd and rs, subtracts them and stores the answer in rd

SUBI rd, imm

23	Rd	Immediate
5	3	8

Takes Rd and subtracts the sign extended immediate value. The result is stored in Rd.

Shifting:

SL rd, imm

2	Rd	Immediate
5	3	8

Takes the immediate value and shifts rd left by the number of bits specified by the immediate. Only the lowest 4 bits of the Immediate will be used because it is only logical to shift the register a maximum of the total number of bits in the register. Using only 4 bits will ensure that Rd can only be shifted by a maximum of 16 places.

SR rd, imm

19	Rd	Immediate
5	3	8

Takes the immediate value and shifts rd right by the number of bits specified by the immediate. Only the lowest 4 bits of the Immediate will be used because it is only logical to shift the register a maximum of the total number of bits in the register. Using only 4 bits will ensure that Rd can only be shifted by a maximum of 16 places.

Conditional Branch:

SLT rt, rd, rs

3	Rd	Rs	Rt	0
5	3	3	3	2

Tests if rd is less than rs, if true, sets rt equal to one, else sets rt to zero

BEQ rd, rs, label

4	Rd	Rs	Label
5	3	3	5

If rd equals rs, goes to the instruction that is an unsigned number of instructions ahead of the next instruction

BNE rd, rs, label

5	Rd	Rs	Label
5	3	3	5

If rd does not equals rs, goes to the instruction that is an unsigned number of instructions ahead of the next instruction

Note:
In order to get around the problem of only being able to jump 32 instructions forward for beq and bne, follow these directions:

1. Switch the logic of the instructions (if it is beq, change it to bne).
2. Load the address of the old Label into a temporary register.
3. The Label of the new beq/bne will be 2.
4. Finally, use the jr command to jump to the original Label address that is stored in the temporary register.

An Example:

Beq rd, rs, 0x0011 0010 becomes:
Bne rd, rs, 3
Lui $g2, 0x0011
Ori $g2, 0x0010
Jr $g2

Unconditional Jump:

JAL label

6	Label
5	11

Jumps to label and stores return address in $ra. See Note about Label for the j instruction below

J label

7	Label
5	11

Jumps to label.
Note: The Label stored in the instruction is bits 11 through 1 of the address of the instruction. The least significant bit is ignored because it is always zero. The most significant 4 bits of the address will come from the PC. This means that if there is a difference between the most significant 4 bits of the PC and the Label, then the jr instruction will need to be used.

JR rd

8	Rd	0	0	0
5	3	3	3	2

Changes PC to the instruction address stored in rd

Data Transfer:

ORI rd, imm

9	Rd	Immediate
5	3	8

Takes an 8-bit imm and performs an or with the lower 8 bits of rd and stores the result in rd.

LUI rd, imm

10	Rd	immediate
5	3	8

Takes an 8bit imm and stores it in the 8 greatest significant bits of the rd and sets the 8 least significant bits to zero.

OR rd, rs

20	Rd	Rs	0	0
5	3	3	3	2

Takes the contents of rd and logical or's it with the contents of rs. The result is stored in rd.

AND rd, rs

21	Rd	Rs	0	0
5	3	3	3	2

Takes the contents of rd and logical and's it with the contents of rs. The result is stored in rd.

SW rd, rs

11	Rd	Rs	0	0
5	3	3	3	2

Stores the word contained in rd to memory address rs

LW rd, rs

12	Rd	Rs	0	0
5	3	3	3	2

Loads the word at address rs into rd

Interrupts and Exceptions

Assert:

ASSIGN IAR, IMM

13	IAR	Immediate
5	3	8

The lower 8-bits of the address of the beginning of the interrupt code will be stored into the correct special purpose register determined by IAR. IAR can be IAR0 or IAR1. The upper 8-bits of the address of the instruction will come from the current value of PC.

Note:
If the PC and the address of the beginning of the interrupt have different most significant 8 bits,

ASSIGNR IAR, Rd

18	IAR	Rd	0	0
5	3	3	3	2

Takes the value stored in Rd, which is the address to the beginning of an interrupt handling. This allows a 16-bit address to be written to the IAR. In order to do this, follow these commands:

```
Lui     Rd, 0x00
Ori     Rd, 0x00
ASSIGNR IAR, Rd
```

Mask:

TOGGLEI

14	0	0	0	0
5	3	3	3	2

Toggles the Enable Interrupt

Return:

ENDI

15	0	0	0	0
5	3	3	3	2

Return from an interrupted instruction and set the EI

Ports

GetPort:

GETPORT PORT, rd

16	Port	rd	0	0
5	3	3	3	2

Puts the current value of the PORT into the register rd.

SetPort:

SETPORT PORT, rs

17	Port	rd	0	0
5	3	3	3	2

Puts the current value of rs into the port value. The input ports, IN0 and IN1, can not be written to in any circumstances. The only valid Port for this command is OUT0, but this must be specified in case more output ports are added in the future.

Note: Port numbers are as follows: IN0 is 000, IN1 is 001, and OUT0 is 011

Registers Conventions

Register Name	Register Code	Usage
$ra	0	Return address
$g1	1	Temporary(not preserved across call)
$g2	2	Argument 1
$g3	3	Argument 2
$g4	4	Temporary(not preserved across call)
$g5	5	Temporary(not preserved across call)
$g6	6	Return 1
$g7	7	Temporary(not preserved across call)

Sample Program

```
Main:
        Jal     point1
        exit
Point1:
        Add     $g2, $ra
        Jal     point2
        Sub     $ra, $ra
        Add     $ra, $g2
        Jr      $ra

Point2:
        Jr      $ra
```

```
#Main:
#00000:
#       Jal     point1
00110 000 0000 0010

#00010
#       exit

#Point1:
#00100
#       Add     $g2, $ra
00000 010 000 000 00

#00110
#       Jal     point2
00110 000 0000 0111

#01000
#       Sub     $ra, $ra
00001 000 000 000 00

#01010
#       Add     $ra, $g2
00000 000 010 000 00

#01100
#       Jr      $ra
01000 000 000 000 00

#Point2:
#01110
#       Jr      $ra
01000 000 000 000 00
```