Assembler Instructions

Example Use

Any instruction can be preceded by a label. Please see example_assembly_text.asm for a more detailed example with offsets and data. Here is an example of a few lines of assembly code:

```
COUNT  .INT  #1
INDEX  .INT  #0
NEXT   .INT  #5
R      .BYT 'R'
A      .BYT 'A'
Y      .BYT 'Y'
       LDR R0, COUNT
       LDR R1, INDEX
       ADD R1, R0
       STR R1, INDEX
       TRP #1
BADX   MOV R3, R0
       CMP R3, R5
       BNZ R3, BADX
       TRP #3
       TRP #0
```

Supported directive notation:

- If a .BYT directive is not between 0-255 inclusive, it's an invalid value.
- For .BYT, you must support the apostrophe-notation such as 'a', 'B', '\t', etc.
  - You are encouraged, but not required, to support decimal notation (such as 97, 66, 8 as the decimal representation of 'a', 'B', and '\t' respectively).
  - You are also encouraged, but not required, to support hexadecimal (such as 0x61, 0x42, 0x08 as the hexadecimal representation of 'a', 'B', and '\t' respectively).
- For .INT, you must support decimal numbers from -2147483648 to 2147483647 inclusive.
  - Anything larger or smaller should be considered errors and cause the assembler to stop.
  - You are also encouraged, but not required, to support hexadecimal notation (for example, 0x80000000 to 0x7FFFFFFF as the hexadecimal representation of -2147483648 and 2147483647 respectively.

Base Instructions

These instructions are used for Projects 1-4.

Additional Guidance for Project 3:

Your ASM ***must*** GET/SET the PC properly

1. Only use ***MOV***, to **GET** the PC and copy it into another register (to calculate a return address). For example:

```
MOV R1, PC
ADD R1, sizeof_instruction * instruction_count
```

2. Only use ***Jump/Branch*** instructions to **SET** the PC

```
BNZ ENDWHILE
```

3. ***Please DO NOT use LDA*** (Load Effective Address) to ***calculate and set a new PC***. *Only use LDA to load a pointer to a memory address into a register!* LDA must NOT be used to get the address of an instruction. For example:

```
LDA R1, ARR
```

4. Do ***not*** use LDA for getting an address of an instruction. Please use the jump and branch instructions instead (see rule #2 above):

~~LDA R1, ENDWHILE~~
~~JMR R1~~

Jump Instructions

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
1	JMP	Branch to Label	Label	1
2	JMR	Branch to address in source register	RS (Source Register)	2
2	BNZ	Branch to Label if source register is not zero	RS, Label	3
2	BGT	Branch to Label if source register is greater than zero	RS, Label	4
2	BLT	Branch to Label if source register is less than zero	RS, Label	5
2	BRZ	Branch to Label if source register is zero	RS, Label	6

Move Instructions

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
1	MOV	Move data from source register to destination register	RD (DestinationRegister), RS	7
(optional)	MOVI	Move immediate value into register.	RD, IMM	31
2	LDA	Load the Address of the label into the RD register. This instruction should ONLY work if the label is associated with a DIRECTIVE.	RD, Label	8
2	STR	Store data into Mem from source register	RS, Label	9
1	LDR	Load destination register with data from Mem	RD, Label	10
2	STB	Store byte into Mem from source register	RS, Label	11
1	LDB	Load destination register with byte from Mem	RD, Label	12

Logical Instructions

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
3	AND	Perform a boolean AND operation, result in destination register. This is a logical AND ***not a bitwise AND***.	RD, RS	18
3	OR	Perform a boolean OR operation, result in destination register. This is a logical OR ***not a bitwise OR***.	RD, RS	19

Compare Instructions

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
2	CMP	Set destination register to zero if destination is equal to source; Set destination register to greater than zero if destination is greater than source; Set destination register to less than zero if destination is less than source.	RD, RS	20
(optional)	CMPI	Set destination register to zero if destination is equal to immediate value; Set destination register to greater than zero if destination is greater than immediate value; Set destination register to less than zero if destination is less than immediate value.	RD, IMM	32

Arithmetic Instructions

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
1	ADD	Add source register to destination register, result in destination register	RD, RS	13
3	ADI	Add immediate data to destination register. Use negative number as subtract-immediate.	RD, ***IMM*** (Immediate constant – value stored in the instruction's operand)	14
1	SUB	Subtract source register from destination register, result in destination register	RD, RS	15
1	MUL	Multiply source register by destination register, result in destination register	RD, RS	16
(optional)	MULI	Multiply immediate to destination register.	RD, IMM	33
1	DIV	Divide destination register by source register, result in destination register	RD, RS	17
(optional)	DIVI	Divide destination register by immediate value	RD, IMM	34

Traps

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
1/2	TRP	Execute an I/O trap routine (a type of operating system or library routine) using register R3. IMM Values 1, write integer to standard out 2, read an integer from standard in 3, write single character to standard out 4, read a single character from standard in Read or write a value from register R3.	***IMM***	21
1	TRP	Execute STOP trap routine. 0, stop program	***IMM***	21
N/A	TRP	DEBUG (OPTIONAL) IMM Value 99	***IMM***	21

Directives

Project # required in	***Directive***	***Description***

1	.INT value	Allocate space for an integer. For example:
		`MONTH    .INT #12`
		`DAY      .INT #9`
		`YEAR     .INT #2012`
		`STUFF    .INT #9`
		`         .INT #-17`
		`         .INT #42`

1	.BYT value	Allocate space for an byte. For example:
		`TWELVE   .BYT #12`
		`PLANETS  .BYT #9`
		`H        .BYT 'H'`
		`i        .BYT 'I'`
		`         .BYT '!'`
		`         .BYT 0x2A`

Registers

Project # required in	*Register*	*Description*	*Value*
1	R[0…15]	General purpose integer registers named R0 through R15	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (0xA), 11 (0xB), 12 (0xC), 13 (0xD), 14 (0xE), 15 (0xF)
1	PC	Program Counter, can't move a value into this register from a MOV instruction but you can copy its value to anotherregister.	16
3	SL	Stack limit (SP with a lower address implies stack overflow)	17
3	SB	Stack Bottom (SP with a higher address implies stack underflow)	18
3	SP	Top of Stack (ToS) aka Stack Pointer. During function return (after copying FP into SP and popping PFP into FP) the SP points to the return address. Be sure to copy the return value ***before*** invoking another function. Before function call, activation record contains, return address, PFP, and function invoking parameters. At the beginning of the function call, allocate other local and temp variables.	19
3	FP	FP current activation record's Frame pointer. Used to address parameters, local vars, and temp vars. Remember since stack grows from highest to lowest memory address that those variables will be FP – (1+Var#)*4. FP during function invocation originally points to the return address.	20

Register Indirect Addressing Instructions

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
2	STR	Store data at register location from source register	RS, RG	22
2	LDR	Load destination register with data at register location	RD, RG	23
2	STB	Store byte at register location from source register	RS, RG	24
2	LDB	Load destination register with byte at register location	RD, RG	25

Multi-threading – Project 4*

Project # required in	***Op Code***	***Description***	***Operands***	***Value***
4	RUN	Create a new thread	REG, LBL	26
4	BLK	Wait for all other threads to terminate before continuing	N/A	27
4	END	Terminate the execution of a non-main thread	N/A	28
4	LCK	Implement a blocking mutex lock	LABEL	29
4	ULK	Remove the lock from a mutex (only on mutexes it locked)	LABEL	30

* More specifics on how these instructions should work are included in project 4.