C335 - Simulated CPU Project

The virtual CPU will be implemented with a 64K memory. It will be organized with sixteen 32-bit registers, an instruction pointer register, and the following instruction set:

Operation	Operands	Comments	Operation Field (hexadecimal)
# robot instruction formats:			
rinit		# initialize communication	00
rsens		# read robot sensors (into registers)	01
rspeed	Rright,Rleft	# set robot motor speeds	02
rspeed	$immed,$immed	# set robot motor speeds	03
# load instruction formats:			
ld	Rd,Rs	# register to register	04
ld	Rd,$immed	# immediate to register	05
ld	Rd,memaddr	# memory address to register	06
ld	Rd,memaddr[Rs]	# memory addr + register to register	07
ld	Rd,*Rs	# register indirect to register	08
# store instruction formats:			
st	Rd,memaddr	# register to memory address	09
st	Rd,memaddr[Rs]	# register to memory addr + register	0A
st	Rd,*Rs	# register to register indirect	0B
# and instruction formats:			
and	Rd,Rs	# register to register	0C
and	Rd,$immed	# immediate to register	0D
and	Rd,memaddr	# memory address to register	0E
and	Rd,memaddr[Rs]	# memory addr + register to register	0F
and	Rd,*Rs	# register indirect to register	10
# or instruction formats:			
or	Rd,Rs	# register to register	11
or	Rd,$immed	# immediate to register	12
or	Rd,memaddr	# memory address to register	13
or	Rd,memaddr[Rs]	# memory addr + register to register	14
or	Rd,*Rs	# register indirect to register	15
# eor instruction formats:			
eor	Rd,Rs	# register to register	16
eor	Rd,$immed	# immediate to register	17
eor	Rd,memaddr	# memory address to register	18
eor	Rd,memaddr[Rs]	# memory addr + register to register	19
eor	Rd,*Rs	# register indirect to register	1A
# not instruction format:			
not	Rd	# register	1B

```
# jump instruction formats:
      jmp   memaddr              # jump to memaddr                        1C
      jgt   Rd,Rs,memaddr        # jump to memaddr if Rd > Rs             1D
      jlt   Rd,Rs,memaddr        # jump to memaddr if Rd < Rs             1E
      jeq   Rd,Rs,memaddr        # jump to memaddr if Rd = Rs             1F
      jez   Rd,memaddr           # jump to memaddr if Rd = 0              20
      jgt   Rd,$immed,memaddr    # jump to memaddr if Rd > $immed         21
      jlt   Rd,$immed,memaddr    # jump to memaddr if Rd < $immed         22
      jeq   Rd,$immed,memaddr    # jump to memaddr if Rd = $immed         23
      jal   Rd,memaddr           # jump and link to memaddr               24
      jmp   *Rd                  # jump register indirect                 25


# add instruction formats:
      add   Rd,Rs                # register to register                   26
      add   Rd,$immed            # immediate to register                  27
      add   Rd,memaddr           # memory address to register             28
      add   Rd,memaddr[Rs]       # memory addr + register to register     29
      add   Rd,*Rs               # register indirect to register          2A
```

The instruction formats above will be translated into machine code as follows:

Instruction Format	16 bits			Notes
	15		0	
op	op	0	0	no operands
op Rd	op	Rd	0	Rs - unspecified
op *Rd	op	Rd	0	Rs - unspecified
op memaddr	op	0	0	Rd - unspecified
	memory reference			Rs - unspecified
op Rd,Rs	op	Rd	Rs	
op Rd,*Rs	op	Rd	Rs	
op Rd,$immed	op	Rd	0	Rs - unspecified
	immediate			immediate - 32-bit,
	immediate			little endian on Pentium
op Rd,memaddr[Rs]	op	Rd	Rs	Rs - index to add to memory
	memory reference			before dereferencing
op Rd,memaddr	op	Rd	0	Rs - unspecified
	memory reference			memory - 16-bit index value
op $immed,$immed	op	0	0	Rd - unspecified
	immediate			Rs - unspecified
	immediate			
	immediate			
	immediate			
op Rd,Rs,memaddr	op	Rd	Rs	
	memory reference			
op Rd,$immed,memaddr	op	Rd	0	Rs - unspecified
	memory reference			
	immediate			
	immediate			

C335 - Simulated CPU Project

A Python program, trans.py, will translate simulated assembly instructions into a machine code file that can be inserted into the simulated CPU memory data array.

To run the translation program, enter the command:

./trans.py [*input filename*] [*output filename*]

input filename The file containing the simulated CPU assembly instructions. If no input filename is specified, trans.py will use the default name 'input.s'. You must specify an input filename if you specify an output filename.

output filename The file that will contain the simulated CPU machine code. If no output filename is specified, trans.py will use the default name 'output.s'. If the output file already exists, it will be overwritten.

Program input:

Each line of code in the input file can consist of up to three fields separated by one or more blank spaces. Any combination of the three fields can be specified in one input line but they must be in the order specified:

[*label*:] [*instruction*] [#*comments*]

[*label*:] The label field is optional but if present, it must be the first field on the line and it must end with a ':'. It may be placed on a line by itself.

[*instruction*] The valid formats for the instruction field are defined in the virtual instruction set list. It is made up of two parts:

operation [*operand*]

The operation field is separated from the operand by one or more blanks. Items in the operand field are delimited by ','. No spaces should appear within the operand field.

Registers specified with the character 'R' followed by one hex digit, 0-9, a-f or A-F. Immediate values are specified beginning with the character "$".

Examples of instruction fields:

```
rinit
ld     R2,Rb
and   R5,label1[R7]
not    R4
jmp   label3+12
```

The instruction field may also be a valid assembly directive statement from the table shown below. Space will be reserved for the defined fields and the

statements will be passed to the output file.

Examples of instruction fields with directive statements:

```
.space   12
.long    1000,4*8,512
.string  "character string field"
```

Directive Statement	Data Type	Notes
.equ	static data symbol	Value cannot be changed.
		Must specify a label field.
.space	number of bytes	
.byte	byte value,[byte value],…	The amount of memory
		reserved depends on the
.word	16-bit integer,[16-bit integer],…	type of data that is defined
		(byte, word, or long)
.long	32-bit integer,[32-bit integer],…	and the number of items
		in the list.
.string	text string	
.ascii	text string	

#comments If present, the comments field must be the last field specified on the line. It can be the only field specified on a line. It must begin with a '#' and it extends from the '#' to the end of the line. It is solely to provide program documentation within the input file of program code. It is not passed to the output file.

C335 - Simulated CPU Project

The table below shows examples of:

- The mnemonic instruction formats for the five addressing modes used by the "load" instruction in the simulated assembly language.
- The layout of the machine code instruction that will be generated.
- The Python assembler output file in the format of assembly directives that can be copied into the simulated CPU memory area.
- The corresponding hexadecimal machine code generated when the CPU simulator program is processed by the GNU Assembler (GAS).

Mnemonic Instruction	Machine Code Format	Notes	Python Output		GAS Output		
					line	offset	instruction
	15 0						
ld R2,R4	04 | Rd | Rs	d = destination register number 0-f s = source register number 0-f	.byte	0x04,0x24	1	0000	0424
ld RA,$1223	05 | Rd | 0	Rs - unspecified	.byte	0x05,0xA0	2	0002	05A0
	immediate	immediate - 32-bit,	.long	1223	3	0004	c7040000
	immediate	little endian on Pentium					
ld Rd,notR9	06 | Rd | 0	Rs - unspecified	.byte	0x06,0xd0	4	0008	06D0
	memory reference	value - 16-bit index into memory	.word	22	5	000a	1600
ld Rf,6984	06 | Rd | 0	Rs - unspecified	.byte	0x06,0xf0	6	000c	06F0
	memory reference	value - 16-bit index into memory	.word	6984	7	000e	481b
ld R2,24[R9]	07 | Rd | Rs	value - 16-bit index	.byte	0x07,0x29	8	0010	0729
	memory reference	to be added to Rs	.word	24	9	0012	1800
ld R1,*Rf	08 | Rd | Rs		.byte	0x08,0x1f	10	0014	081F
notR9:							
not R9	1B | Rd | 0	Rs - unspecified	.byte	0x18,0x90	11	0016	1890