UVU 4380 Instruction Set Architecture

Overview

Instruction Size: Fixed at 8 Bytes (64 bits)

1 Byte Operation	1 Byte Reg Operand 1	1 Byte Reg Operand 2	1 Byte Reg Operand 3	4 Byte Immediate Value

Registers (22):

Name	Encoding	Description (all registers are 32 bits in size)
R0 – R15	0 - 15	General purpose registers, **initialized to 0 at startup**
PC	16	Program Counter, **initialized to the value of first instruction in memory**
SL	17	Stack Lower limit (lowest legal address available to the stack)
SB	18	Stack Bottom (highest address)
SP	19	Stack Pointer (latest allocated byte on the stack, grows downward)
FP	20	Frame Pointer (points to the first word beneath the return address)
HP	21	Heap Pointer (initially set to SL, grows upward))

Memory: 32 bit address space; Byte addressable; Little-endian; User specifiable; Defaults to 2^17 (131,072) bytes of addressable memory.

Instructions

Note: Several operands are marked as DC, meaning "Don't Care". This means that while the operand is required - its specific value doesn't matter to execution of the instruction.

Jump Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
1	JMP	DC*	DC*	DC*	Address**	Jump to Address
2	JMR	RS	DC*	DC*	DC*	Update PC to value in RS
3	BNZ	RS	DC*	DC*	Address**	Update PC to Address if RS != 0
4	BGT	RS	DC*	DC*	Address**	Update PC to Address if RS > 0
5	BLT	RS	DC*	DC*	Address**	Update PC to Address if RS < 0
6	BRZ	RS	DC*	DC*	Address**	Update PC to Address if RS = 0

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.
**When represented in an assembly file the Address shall be represented as a label. This label must be resolved to a 32 bit unsigned integer value in bytecode.

Move Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
7	MOV	RD	RS	DC*	DC*	Move contents of RS to RD
8	MOVI	RD	DC*	DC*	Imm*	Move Imm value into RD
9	LDA	RD	DC*	DC*	Address**	Load address into RD
10	STR	RS	DC*	DC*	Address**	Store integer in RS at Address
11	LDR	RD	DC*	DC*	Address**	Load integer at Address to RD
12	STB	RS	DC*	DC*	Address**	Store least significant byte in RS at Address
13	LDB	RD	DC*	DC*	Address**	Load byte at Address to RD
14	ISTR	RS	RG	DC*	DC*	Store integer in RS at address in RG
15	ILDR	RD	RG	DC*	DC*	Load integer at address in RG into RD
16	ISTB	RS	RG	DC*	DC*	Store byte in RS at address in RG
17	ILDB	RD	RG	DC*	DC*	Load byte at address in RG into RD

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.
**When represented in an assembly file the Address shall be represented as a label. This label must be resolved to a 32 bit unsigned integer value in bytecode.

Arithmetic Operations:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
18	ADD	RD	RS1	RS2	DC*	Add RS1 to RS2, store result in RD
19	ADDI	RD	RS1	DC*	Imm**	Add Imm to RS1, store result in RD
20	SUB	RD	RS1	RS2	DC*	Subtract RS2 from RS1, store result in RD
21	SUBI	RD	RS1	DC*	Imm**	Subtract Imm* from RS1, store result in RD
22	MUL	RD	RS1	RS2	DC*	Multiply RS1 by RS2, store result in RD
23	MULI	RD	RS1	DC*	Imm**	Multiply RS1 by Imm, store the result in RD
24	DIV	RD	RS1	RS2	DC*	Perform unsigned integer division RS1/ RS2. Store quotient in RD. Division by zero shall result in an emulator error
25	SDIV	RD	RS1	RS2	DC*	Store result of signed division RS1 / RS2 in RD. Division by zero shall result in an emulator error
26	DIVI	RD	RS1	DC*	Imm**	Divide RS1 by Imm (signed), result in RD. Division by zero shall result in an emulator error

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.
** When represented in an assembly file the Imm value shall be represented as a numeric literal.

Logical Instructions: (NOT BITWISE)

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
27	AND	RD	RS1	RS2	DC*	Perform RS1 && RS2, store result in RD
28	OR	RD	RS2	RS2	DC*	Perform RS1 || RS2, store result in RD

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

Compare Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
29	CMP	RD	RS1	RS2	DC**	Set RD = 0 if RS1 == RS2 | Set RD = 1 if RS1 > RS2 | Set RD = -1 if RS1 < RS2
30	CMPI	RD	RS1	DC**	Imm*	Set RD = 0 if RS1 == Imm | Set RD = 1 if RS1 > Imm | Set RD = -1 if RS1 < Imm

* When represented in an assembly file the Imm value shall be represented as a numeric or character literal.

**This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

Traps/Interrupts:

Value	Operator	Operand 1*	Operand 2**	Operand 3**	Immediate Value**	Description
31	TRP	#0	DC**	DC**	DC**	Executes the STOP/Exit routine
31	TRP	#1	DC**	DC**	DC**	Write int in R3 to stdout (console)
31	TRP	#2	DC**	DC**	DC**	Read an integer into R3 from stdin
31	TRP	#3	DC**	DC**	DC**	Write char in R3 to stdout
31	TRP	#4	DC**	DC**	DC**	Read a char into R3 from stdin
31	TRP	#5	DC**	DC**	DC**	Write the null-terminated Pascal-style string whose starting address is in R3 to stdout.
31	TRP	#6	DC**	DC**	DC**	Read a newline terminated string from stdin and store it in memory as a null-terminated Pascal-style string whose starting address is in R3. (Do not store the newline)
31	TRP	#98	DC**	DC**	DC**	Print all register contents to stdout

*Note: The leading '#' is required in 4380 assembly programs; only the following/trailing numeric value is represented in 4380 bytecode.

**DC = "Don't Care". These operands shall be omitted in 4380 assembly instructions but are required in 4380 bytecode.

Heap Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
32	ALCI	RD	DC***	DC***	Imm*	Allocate Imm bytes of space on the heap (and increment HP accordingly). The Imm value is a 4-byte unsigned integer. Store the initial heap pointer in RD.
33	ALLC	RD	DC***	DC***	Address**	Allocate a number of bytes on the heap according to the value of the 4-byte unsigned integer stored at Address (and increment HP accordingly). Store the initial heap pointer in RD.
34	IALLC	RD	RS1	DC***	DC***	Indirectly allocate a number of bytes on the heap according to the value of the 4-byte unsigned integer at the memory address stored in RS1 (and increment HP accordingly). Store the initial heap pointer in RD.

* When represented in an assembly file the Imm value shall be represented as a numeric literal.
** When represented in an assembly file the Address shall be represented as a label in the data section.
*** DC = "Don't Care". These operands shall be omitted in 4380 assembly instructions but are required in 4380 bytecode.

Procedure/Stack Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
35	PSHR	RS	DC*	DC*	DC*	Set SP = SP - 4; Place the word in RS onto the stack
36	PSHB	RS	DC*	DC*	DC*	Set SP = SP - 1; Place the least significant byte in RS onto the stack
37	POPR	RD	DC*	DC*	DC*	Place the word on top of the stack into RD, update SP = SP + 4
38	POPB	RD	DC*	DC*	DC*	Place the byte on top of the stack into RD, update SP = SP + 1
39	CALL	Address**	DC*	DC*	DC*	Push PC onto stack, update PC to Address
40	RET	DC*	DC*	DC*	DC*	Pop stack into PC

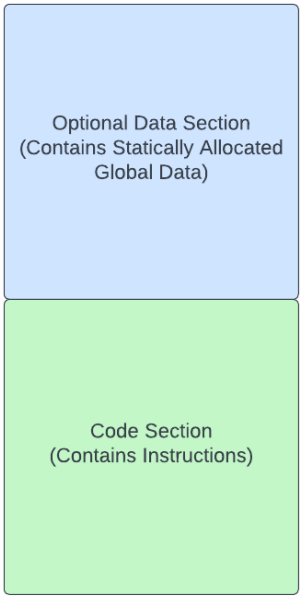
*DC = "Don't Care". This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

** When represented in an assembly file the Address shall be represented as a label.

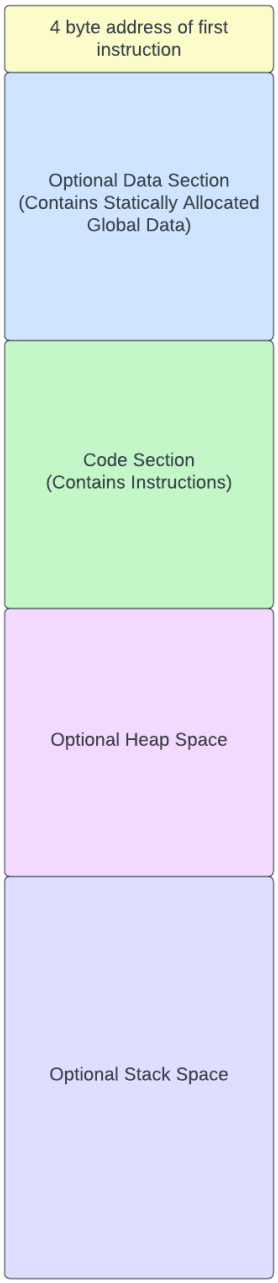
4380 Assembly Language Programming

File Requirements

4380 assembly programs must be contained in text files with the '.asm' file extension (e.g. gradebook.asm). Each file is comprised of up to two sections - an optional Data Section (which contains the static and global variable definitions for the program), followed by a Code Section containing the program instructions. The Data Section (if present) must precede the Code Section. Both sections are described in greater detail below.



This file format coincides with the following convention for programmable memory layout of the 4380 at runtime:



Line Format

Four valid line types may occur within a 4380 assembly file:

1. Blank lines: Containing only whitespace characters. These will be ignored by the assembler.
2. Comment lines: Containing zero or more whitespace characters followed by a valid comment. These will be ignored by the assembler.
3. Directive lines: Beginning with a label or whitespace character(s) followed by a valid directive.

4. Instruction lines: Beginning with a label or whitespace character(s) followed by a valid instruction.

Note that both directive and instruction lines begin with a label or whitespace, meaning that if a valid line begins with an alphanumeric character, that character is the beginning of a label.

Programmer Comments

4380 assembly files may contain programmer comments throughout the file. Comments begin with the ';' character and extend to the end of a line (much like C-style comments). All text following the ';' symbol shall be ignored by the 4380 Assembler.

```
;This example comment occupies an entire line.
A_LABEL     .BYT  #245;This comment occupies part of a line
```

Labels

Labels may be used to mark important locations within both the Data and Code sections of a 4380 assembly program (these sections are discussed below). Label definitions appear as the optional first elements of Directive Lines and Instruction Lines (they must not appear on a line by themselves). They must be left justified (no preceding whitespace characters). Label names must begin with an alphanumeric character and may contain alphanumeric characters, underscores, and the '$' character.

For example:

```
labelName9
NUM_APPLES
function$
```

...are all valid labels, while:

```
_bad
```

...is invalid.

Note: Use of the '$' character in manually written 4380 assembly labels is discouraged, as limiting this character to compiler generated labels can prevent naming collisions between programmer and compiler generated labels/identifiers.

Label names can be used as operands for instructions in the Code Section of a 4380 assembly.

Literal Data Representation

Both directives and instructions may make use of literal data within a 4380 assembly file. The supported numeric and character literals are as follows:

- Numeric: Signed integer value represented as a base 10 number preceded by the '#' character. Examples: #45, #2, #-67
- Character: Apostrophe delineated ascii character (including escape sequences for special characters), or a character's equivalent ascii code expressed as a decimal value. Examples: 't', 'Y',

'6', '#', '\n', #83 (83 is the ascii code for the character 'S') The following seven escape sequences must be supported by the 4380 assembler:

- \t (tab)
- \\ (backslash)
- \n (newline)
- \' (single quote)
- \" (double quote)
- \r (carriage return)
- \b (backspace)

Data Section and Directives

The Data Section of a 4380 assembly contains assembler directives that define static and global variables. The Data Section implicitly begins with the first line of an assembly file and continues until the first line containing an assembly instruction.

Directives can only be used within the Data Section. The presence of one of these directives in the Code Section shall result in an assembler error. All directives begin with a period, making them easy to identify when parsing a 4380 assembly file. The supported directives for the Mark 1 assembler are as follows:

Directive	Operand	Examples	Behavior
.INT	Optional signed decimal value in the range -2147483648 to 2147483647 inclusive.	.INT #45 .INT #-12 .INT #2147483647	Allocates memory in place for a 4 byte integer and initializes it with the optional operand value. If no operand is provided the value is initialized to 0.
.BYT	Optional unsigned decimal value in the range 0 to 255 inclusive, OR apostrophe delineated ascii character (including escape sequences for special characters)	.BYT #45 .BYT 'a' .BYT '\n'	Allocates 1 byte of memory in place and initializes it with the optional decimal value or ascii code. If no operand is provided the value is initialized to 0.
.BTS	Required unsigned decimal value. This value is the number of bytes to be allocated.	.BTS #25 .BTS #5 a_label .BTS #20	Allocates the specified number of bytes in place and initializes them all to 0. If an optional label is present the label shall be associated with the address of the first byte allocated.
.STR	Required double quote delimited string OR a numeric literal. 255 shall be the maximum length of the string and the maximum value of the numeric literal.	.STR "Example!" name .STR "Fred" .STR #40 .STR #255	Allocates a number of bytes equal to the length of the string + 2. The first byte is initialized to the length of the string, and the last byte to 0 (null character). The

			remaining bytes (in the middle) are initialized to the ascii values for the characters in the string. **Note:** escape sequences such as \n must be properly handled. OR Allocates a number of bytes equal to the numeric literal + 2. The first byte is initialized to the value of the numeric literal and the remaining bytes are initialized to zeros. In both cases if an optional label is present the label shall be associated with the address of the first byte allocated.

Note that the assembler must treat directives as case INsensitive. For example ".int" and ".INT" must be treated the same by the assembler. The Data Section of an assembly may contain as many directives as desired (within the limitations of the emulator's memory) but shall be limited to one directive per line. Each directive may optionally be preceded by a label, which in this context serves as a variable or array name.

```
NUM_APPLES          .INT  #42; An integer variable initialized to 42
CURRENT_MONTH       .BYT ; An unsigned int variable initialized to 0
A_CHAR              .BYT  'A'; A char variable initialized to the
                             ; character 'A'
A_CHAR_ARRAY        .BYT  'H'; The first character of an array of
                             ; characters
                    .BYT  'e'
                    .BYT  'l'
                    .BYT  'l'
                    .BYT  'o'
```

Directive Format
Considering the preceding discussion the format of a 4380 directive can be summarized as follows:
`[optional_label] <.directive> [optional_value][optional_comment]`
Note the presence of required whitespace between the directive and the optional label and value. This whitespace must be comprised of space and/or tab characters.

Code Section

The Code Section of a 4380 assembly file implicitly begins with the first line containing an instruction, which must also be the following instruction:

```
JMP   MAIN
```

Clearly this implies that a 4380 assembly file must contain a MAIN function/procedure that is properly marked with the label 'MAIN'. The absence of such a label and function shall result in an assembler error.
As with the Data Section and directives, the Code Section may contain as many instructions as are desired (within the limitations of the emulator's memory), but shall be limited to one instruction per line.

NOTE: Assembly directives and instructions cannot be interleaved. Directives are limited to the Data Section, and instructions must reside in the Code Section of a file.

Instruction Format
4380 Instructions must be formatted as follows:

`[optional_label] <operator> <operand_list> [optional_comment]`

As with the format for directives, note the required whitespace (which can be comprised of spaces and/or tabs) separating the individual components of an instruction.

An instruction's `operator` is limited to one of the string representations of valid 4380 operators as listed in the instruction tables utilized in this course. The assembler treats operators as case INsensitive. For example "aDd" and "ADD" should both be recognized/accepted by the assembler.

The `operand_list` is a comma separated list of the necessary operands for the operator being invoked. Valid operands include:

- Registers - indicated via the two or three-character register names listed in the register tables utilized in this course. Note, the assembler shall treat register names as case INsensitive. R13 is considered the same as r13.
- Numeric constants - represented as literal values (e.g. #45).
- Labels - as defined in the assembly file and later resolved to addresses by the assembler. Labels are to be treated as case SENSITIVE by the assembler.