CSC 106 - Spring 2016
Pep/8 reference

1 Architecture Summary

The Pep/8 processor uses a 16-bit word size, and has 2^{16} memory locations, each of which is one byte (8 bits) in size. All registers are 16 bits in size, and memory addresses are 16 bits long. When a word in a register is stored to memory, two memory locations are used, since the word size is two bytes and memory locations are one byte in size.

The general purpose registers are the accumulator (`A`) and the index register (`X`). The architecture contains several other registers for special uses, including the program counter (`PC`), which tracks the location of the current instruction in memory, and the stack pointer (`SP`), which is used to track the top of the call stack in memory. The significance of the special purpose registers is covered in an architecture course, such as CSC 230.

2 Instruction Set

The table below describes the subset of the Pep/8 instruction set which will be necessary for CSC 106. A full instruction reference can be found in the documentation for the Pep/8 simulator. In the table, the identifier `VAR` is used as a placeholder for a variable in memory (created with the `.WORD` directive), the identifier `LABEL` is used as a placeholder for a label and the constant 106 is used as a placeholder for any numerical constant (numerical constants can be positive or negative).

Instruction	Description
LOAD/STORE	
`LDA 106, i` `LDX 106, i`	Set register A (`LDA`) or X (`LDX`) to the value 106.
`LDA VAR, d` `LDX VAR, d`	Set register A (`LDA`) or X (`LDX`) to the value at memory address `VAR`.
`STA VAR, d` `STX VAR, d`	Store the value in register A (`LDA`) or X (`LDX`) into memory location `VAR`.
ARITHMETIC	
`ADDA 106, i` `ADDX 106, i`	Add 106 to register A (`ADDA`) or X (`ADDX`).
`ADDA VAR, d` `ADDX VAR, d`	Add the contents of memory address `VAR` to register A (`ADDA`) or X (`ADDX`).
`SUBA 106, i` `SUBX 106, i`	Subtract 106 from register A (`SUBA`) or X (`SUBX`).
`SUBA VAR, d` `SUBX VAR, d`	Subtract the contents of memory address `VAR` from register A (`SUBA`) or X (`SUBX`).
`NEGA` `NEGX`	Negate the value of register A (`NEGA`) or X (`NEGX`)
COMPARISON	
`CPA 106, i` `CPX 106, i`	Compare the value of register A (`CPA`) or X (`CPX`) to the constant 106. The result of the comparison will be saved for future branch instructions.
`CPA VAR, d` `CPX VAR, d`	Compare the value of register A (`CPA`) or X (`CPX`) to the contents of memory address `VAR`. The result of the comparison will be saved for future branch instructions.
BRANCH	
`BR LABEL, i`	Jump to the instruction at `LABEL`.

`BREQ LABEL, i`
`BRNE LABEL, i`
`BRLT LABEL, i`
`BRLE LABEL, i`
`BRGE LABEL, i`
`BRGT LABEL, i`
	Jump to the instruction at `LABEL` only if the last comparison (`CP`) operation was equal (`BREQ`), not equal (`BRNE`), less than (`BRLT`), less than or equal (`BRLE`), greater than or equal (`BRGE`) or greater than (`BRGE`).

INPUT/OUTPUT	
`DECI VAR, d`	Read a base 10 value from the user and store it in memory location `VAR`.
`DECO 106, i`	Output the value 106 in base 10.
`DECO VAR, d`	Output the contents of memory location `VAR` in base 10.
`CHARO 'a', i`	Output the character '`a`'. (The character '`\n`' is useful for ending lines).
`STRO STR, d`	Output the text string at memory location `STR`.
MISC.	
`STOP`	Halt execution
`NOP`	Do nothing and continue to the next instruction. (This is useful as a placeholder operation)

3 Other Assembly Directives

The Pep/8 assembler supports several 'pseudo-opcodes', beginning with the '.' character, for memory management and administrative tasks.

3.1 End of Code

The Pep/8 assembler requires that the `.END` directive be placed at the end of every assembly file, after all of the code (including variable declarations).

3.2 Numerical Variables

To create a named variable which can be used to store numerical values (and can be used with instructions like `ADDA`), use the `.WORD` directive:

```
VAR: .WORD 6
```

The name of the variable is specified before the colon (in the example above, the name is '`VAR`'). The initial value of the variable is specified after the `.WORD` directive (in the example above, the initial value is 6).

3.3 Strings of Text

Strings of text can be created with the `.ASCII` directive. For the purposes of this course, strings of text are only used for output, with the `STRO` instruction.

```
STR:  .ASCII "This is a string of text.\x00"
```

The name of the string is specified before the colon (and functions exactly the same way as any other variable name). The text itself is enclosed in double quotes after the `.ASCII` directive. Note that for `STRO` to work correctly, the string must end with '`\x00`' (the seqeuence '`\x00`' produces a *null terminator*, which signals the end of the string to the `STRO` instruction).