Answers ICAR Software Laboratory : VAX

Question: When reading an instruction from memory what is the advantage of specifying the destination register last? Answer: may be possible to fetch operand data whilst instruction is read (instruction may take a number of cycles to read) i.e. result has not been generated, therefore, destination register is not needed at this time.

Question: Why would an `Addb2` instruction execute faster than an `Addb3` instruction? Answer: may be quicker as instruction length is smaller as less information must be read from memory.

Questions: What is the advantage / disadvantage of an instruction set that allows the numbers of operands to be selected i.e. 2 or 3 operand formats? Answer: don't always need three operands i.e. separate source and destinations, therefore, can save instruction memory by using a smaller instruction.

Question: how many bits will be required to encode this `addl3` instruction? Answer:

```
addl3 15(r2),(r3)[r4], r1      ;3 operand Long word ADD
```

8 : 16 : 4 4 4 4 = 40 bits

Question: If operands can be either register, memory, or constants, what is the maximum and minimum number of bits required to represent this instruction? Answer: the biggest length instruction would be one that uses absolute addressing as a full 32bit address with be required for each operand and result

8 : 32 : 32 : 32 = 104 bits

Task 1

Question: Why is a move long instruction used to load `R0`? Answer: processor uses a 32bit address.

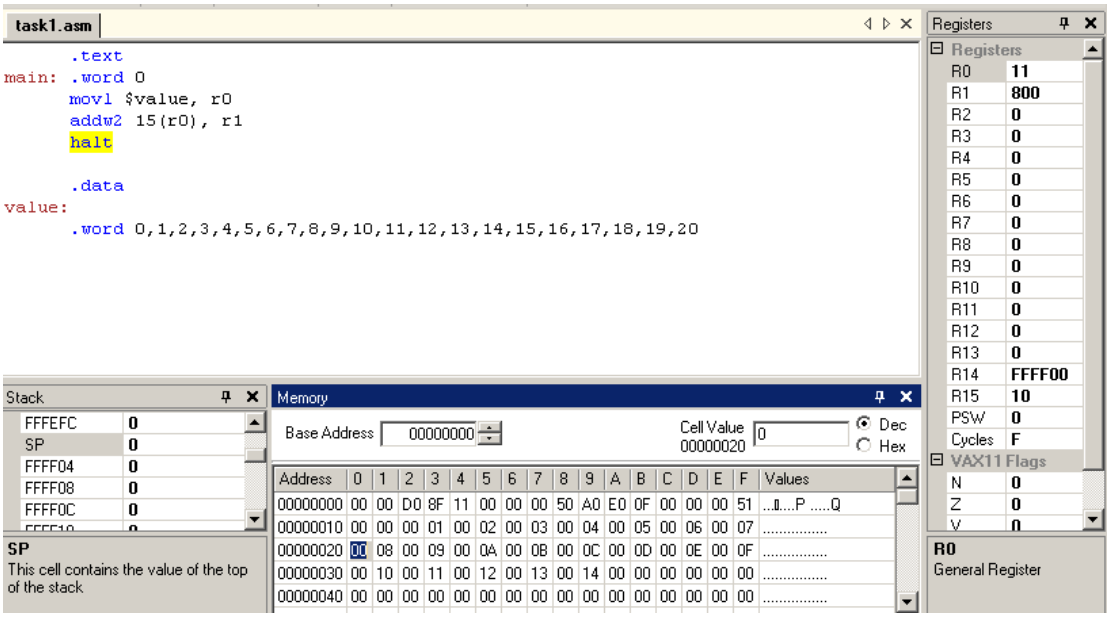
Question: Where and how are the data value 0 – 20 stored in memory? Answer: data starts at memory location 0x11 and is stored as 16bit values i.e. takes up two memory locations. **Update**: running the 'same' code this year the address was 0x0E? Not sure why there would be a difference, will investigate.

Question: What will the values in `R0` and `R1` be when the program finishes, is this result correct? Answer: when first powered up R1=0, therefore, add performs the same function as move / input data from memory.

R0 = 11 R1 = 800

R1 may not be the result you were expecting. This is caused by the offset 15, misaligned access, 17 + 15 = 32 (0x20), accessing the wrong low and high bytes. This is illustrated when the offset is changed to 16 it now correctly loads the value 8 i.e.

reads the low byte from address 0x21 and the high byte from 0x22, a little Endian data format.

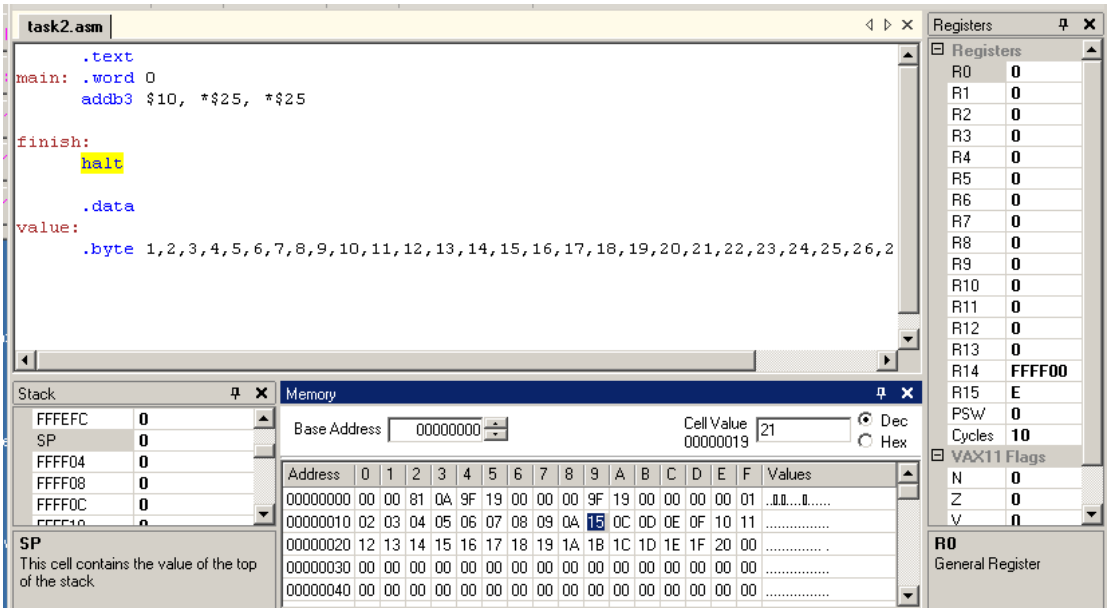


Task 2

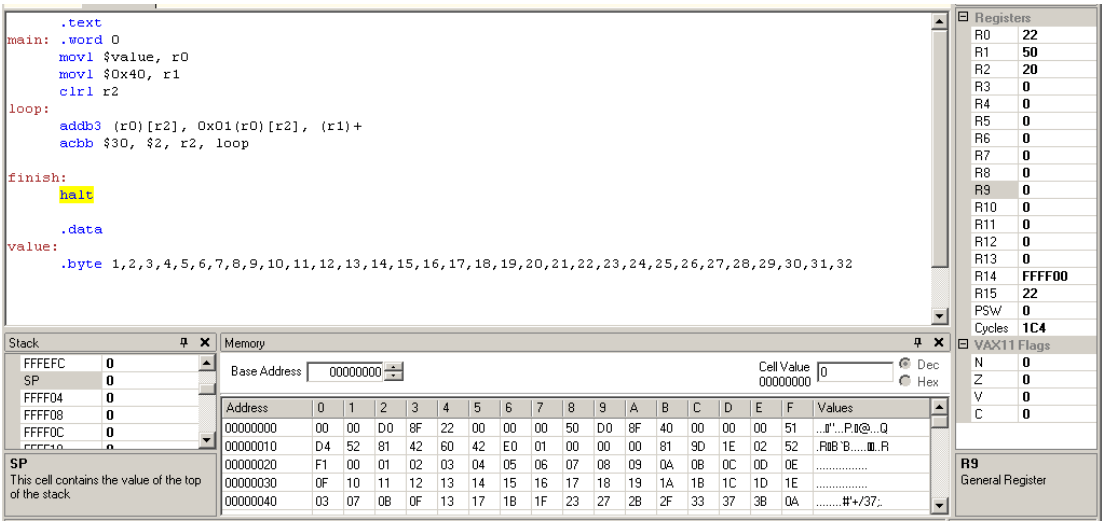
PicoBlaze needs three instructions

Input SF, 19
Add SF, 0A
Output SF, 19

VAX needs only one instruction



Task 3



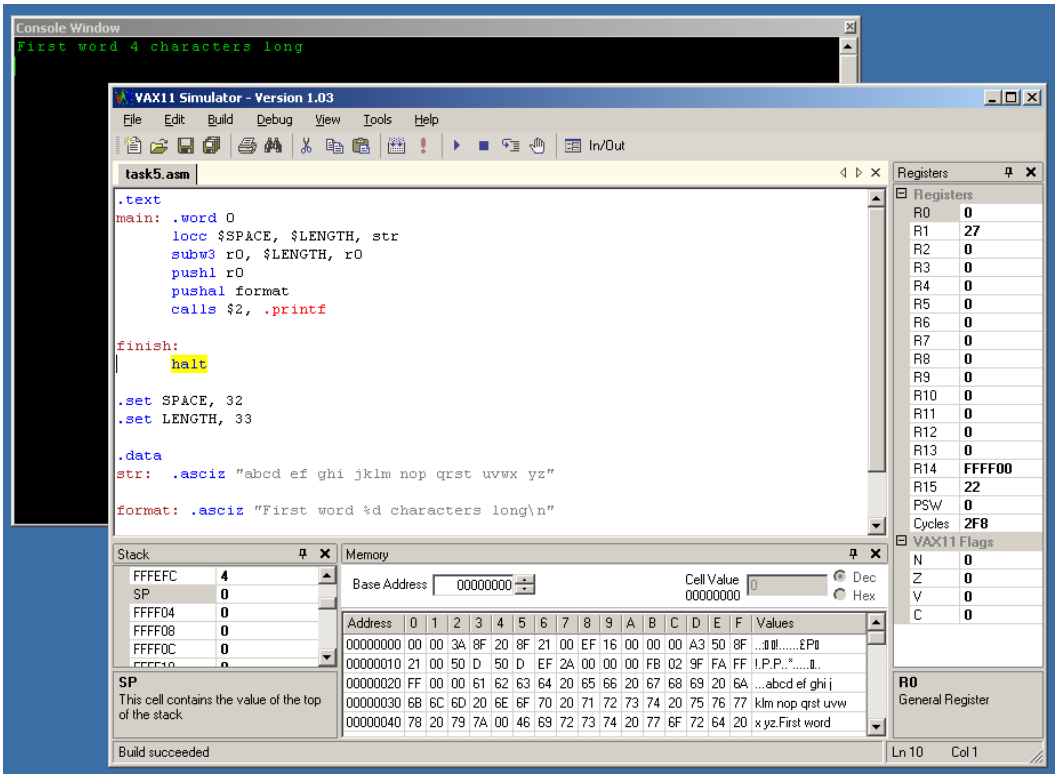
```
        .text
main:   .word 0
        movl $value, r0
        movl $0x40, r1
        clrl r2
loop:
        addb3 (r0)[r2], 0x01(r0)[r2], (r1)+
        acbb $30, $2, r2, loop

finish:
        halt

        .data
value:
        .byte 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
```

VAX : IC = 3 + 2× 16 + 1 = 36
PicoBlaze: IC = 3 + 11 × 16 + 1 = 180

Advantages: less instructions, lower clock speeds, less power (maybe)
Disadvantage: more hardware, increased cost, more power (maybe)

Task 4

Finds the first space character i.e. the end of the first word



Additional Task 1

use MATCHC: find substring within character string, very easy, BUT, can't seem to get this instruction to do what is says it will do, maybe ive misunderstood the instruction or how it should be used. Anyway solution B is below. This is a little bit more complex, I think there should be a nicer solution, definitely a little rushed/hacked together, if anyone comes up with a nicer solution do email me.

```
.text
main: .word 0
      movl $str, r5           #load string pointer
      movl $key, r6           #load key pointer
      movl $STR_LENGTH, r7    #load string length

loop:
      locc $SPACE, r7, (r5)   #find space in string
      subw3 r0, r7, r8        #calc location from start

      cmpw $KEY_LENGTH, r8    #is substring the same size as key
      beql test               #yes, test

update:
      addl3 r5, r8, r5        #move pointer
      addl3 $1, r5, r5
      subl3 r8, r7, r7        #reduce length
      bleq finish
      jmp loop

test:
      cmpc3 $KEY_LENGTH, (r6), (r5) #are the two strings equal?
      beql found                    #yes print position
      jmp update                    #no loop

found:
      pushl r5                      #print address to screen
      pushal message
      calls $2, .printf
      jmp update                    #loop

finish:
      halt

.set SPACE, 0x20
.set STR_LENGTH, 33
.set KEY_LENGTH, 3

.data
str:     .asciz "abcd ef ghi jklm nop qrst uvwx yz"
key:     .asciz "nop"
message: .asciz "Match: %d\n"
```