

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 = 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 will 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.

task1.asm

```

.text
main: .word 0
      movl $value, r0
      addw2 15(r0), r1
      halt

.data
value:
      .word 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20

```

Registers

R0	11
R1	800
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
R13	0
R14	FFFF00
R15	10
PSW	0
Cycles	F

VAX11 Flags

N	0
Z	0
V	0

Stack

FFFEFC	0
SP	0
FFFF04	0
FFFF08	0
FFFF0C	0
FFFF10	0

Memory

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Values
00000000	00	00	D0	8F	11	00	00	50	A0	E0	0F	00	00	00	51	...	P...Q
00000010	00	00	01	00	02	00	03	00	04	00	05	00	06	00	07
00000020	00	08	00	09	00	0A	00	0B	00	0C	00	0D	00	0E	00	0F
00000030	00	10	00	11	00	12	00	13	00	14	00	00	00	00	00	00
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Task 2

PicoBlaze needs three instructions

Input SF, 19

Add SF, 0A

Output SF, 19

VAX needs only one instruction

task2.asm

```

.text
main: .word 0
      addb3 $10, *$25, *$25

finish:
      halt

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

```

Registers

R0	0
R1	0
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0
R9	0
R10	0
R11	0
R12	0
R13	0
R14	FFFF00
R15	E
PSW	0
Cycles	10

VAX11 Flags

N	0
Z	0
V	0

Stack

FFFEFC	0
SP	0
FFFF04	0
FFFF08	0
FFFF0C	0
FFFF10	0

Memory

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Values
00000000	00	00	81	0A	9F	19	00	00	00	9F	19	00	00	00	01
00000010	02	03	04	05	06	07	08	09	0A	15	0C	0D	0E	0F	10	11
00000020	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Task 3

The screenshot shows the VAX11 simulator interface. The main window displays assembly code for a program that calculates the sum of integers from 1 to 32. The code includes instructions like `movl $value, r0`, `movl $0x40, r1`, `clrl r2`, `addb3 (r0)[r2], 0x01(r0)[r2], (r1)+`, `acbb $30, $2, r2, loop`, and `halt`. The `halt` instruction is highlighted in yellow.

The right-hand pane shows the Register File with R15 containing the value 22. The VAX11 Flags pane shows the Carry flag (C) set to 0. The bottom pane shows the Memory window with a table of addresses and values:

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Values
00000000	00	00	D0	8F	22	00	00	00	50	D0	8F	40	00	00	00	51	..#"...P#@.0
00000010	D4	52	81	42	60	42	E0	01	00	00	00	81	9D	1E	02	52	..RIB'B...@.R
00000020	F1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
00000030	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E
00000040	03	07	0B	0F	13	17	1B	1F	23	27	2B	2F	33	37	3B	3F#*/37.

$$\text{VAX : IC} = 3 + 2 \times 16 + 1 = 36$$

$$\text{PicoBlaze: IC} = 3 + 11 \times 16 + 1 = 180$$

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

The screenshot shows the VAX11 simulator interface. The main window displays assembly code for a program that finds the first space character in a string. The code includes instructions like `locc $SPACE, $LENGTH, str`, `subw3 r0, $LENGTH, r0`, `pushl r0`, `pushal format`, and `calls $2, .printf`. The `halt` instruction is highlighted in yellow.

The right-hand pane shows the Register File with R15 containing the value 27. The VAX11 Flags pane shows the Carry flag (C) set to 0. The bottom pane shows the Memory window with a table of addresses and values:

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Values
00000000	00	00	3A	8F	20	8F	21	00	EF	16	00	00	A3	50	8F	..#!.....EPI	
00000010	21	00	50	D	50	D	EF	2A	00	00	FB	02	9F	FA	FF	!.P.P...@.I.	
00000020	FF	00	00	61	62	63	64	20	65	66	20	67	68	69	20	6A	...abcd ef ghij
00000030	6B	6C	6D	20	6E	6F	70	20	71	72	73	74	20	75	76	77	klm nopqrst uvw
00000040	78	20	79	7A	00	46	69	72	73	74	20	77	6F	72	64	20	x yz.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 it 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"
```