—--Addressing modes-—-
Register addressing: Value < Register
Immediate: Value < Data field in instruction
Direct addressing: Value <> memory[Memory Address]
//The instruction directly specifies the address of the operand.
Relative: Value > memory[Base Memory Address + fixed offset]
Indexed:
- Value ‹› memory[Memory Address)
- Memory Address is modified (+|-) for next access
Indirect: (Address refers to memory location with the address): Value < memory [memory[Memory Address)]
// The instruction specifies a register or memory location that contains the address of the operand
Direct addressing is limited in that unless the register used to determine the effective address is modified, the instruction will access the same location each time it is executed. A variation on direct addressing is to modify the register within the load or store instruction itself, allowing access to, for example, array structures. This is referred to as indexed addressing with the effective address being obtained from a register using as an index register.
XM-23, like most other machines that use indexed addressing allows the programmer to increment or decrement the Index register before or after accessing a memory location. This is referred to as pre-increment, pre-decrement, post-increment, and post-decrement.
The format of the LD (load) and ST (store) Instructions using indexed addressing
Sure, here are the definitions for the 24 computer architecture terms:
CISC vs. RISC machines
CISC-Complex Instruction Set Computer:
-Most instructions can generate EA:
ADD mem1,mem2
SUB #4,mem1
-Combinations of registers and memory permitted:
ADD R1,mem2
SUB R1,R2
• RISC – Reduced Instruction Set Computer (ours):
Limited number of instructions can generate EA:
LOAD mem1,Rx
STORE Rx,mem2
General Purpose Register: A register in a CPU that can store data or addresses and be used for various instructions by the processor, not limited to specific functions.
MBR (Memory Buffer Register):A register that holds data read from or written to memory. It acts as a buffer between the CPU and the memory.
Tail Chaining: A technique in interrupt handling where the processor can handle back-to-back interrupts efficiently without returning to the main program between interrupts.
Bit-slice:A design technique where a processor or arithmetic logic unit (ALU) is constructed from modules, each of which processes a slice (usually a few bits) of the full data width.
Directive:An instruction in assembly language programming that provides guidance to the assembler but does not generate machine code, such as specifying data storage, alignment, or code generation options.
Relative Addressing:An addressing mode where the effective address is determined by the index value specified in the instruction, relative to the current program counter (PC).
Write Through:A caching technique where every write operation to the cache is simultaneously written to both the cache and the main memory, ensuring consistency between them.
Subroutine: A sequence of instructions that perform a specific task, packaged as a unit. It can be called from multiple points in a program to perform the task whenever needed.
Index Register:A register used in indexed addressing modes to hold an index value, which is added to a base address to get the effective address of an operand.
Harvard Architecture(ours):A computer architecture with separate memory spaces for instructions and data, allowing simultaneous access to both, which can improve performance.
Instruction Decoder:A part of the CPU that interprets the binary opcode of an instruction and determines the operation to be performed and the operands involved.
Cache Coherence:A consistency protocol ensuring that multiple caches in a system reflect the most recent value of shared data, preventing data inconsistency.
Accumulator:A register in which intermediate arithmetic and logic results are stored in a CPU. It is commonly used in early computer architectures for operations.
Zero-Address Machine:A machine that uses a stack-based architecture where instructions do not require explicit operands because they implicitly use the top values of the stack.
Program Status Word (PSW):A register containing the status flags and control bits of the CPU, reflecting the current state of the processor and influencing its operation.
Exception:An unexpected event or error that occurs during the execution of a program, requiring special handling by the CPU, often through an interrupt or trap mechanism.
Interrupt Vector:A memory location or table containing addresses of interrupt service routines. When an interrupt occurs, the CPU uses the interrupt vector to find the correct service routine.
Symbol Table:A data structure used by assemblers and compilers to keep track of symbols (variable names, function names, etc.) and their corresponding addresses or values.
Program Counter (PC):A register in the CPU that contains the address of the next instruction to be executed in the program sequence.
ISA (Instruction Set Architecture):The part of the computer architecture that specifies the machine code instructions the CPU can execute, along with their format, behavior, and addressing modes.
DMA (Direct Memory Access):A feature that allows peripheral devices to transfer data directly to and from memory without involving the CPU, improving data transfer efficiency.
Cache Line:The smallest unit of data transfer between the cache and main memory. It typically consists of multiple contiguous bytes of memory.
Deterministic Loop:A loop in programming that has a known and predictable number of iterations, often used in real-time systems where timing predictability is crucial.
1000 6FF8 MOVLZ #FF, R0 1002 58B9 LD PC+, R1 1004 FE01 WORD #FE01 1006 4008 ADD R1, R0
^results in r0:x00ff, r:xfe01
WHILE
CMP #0, R0 BEQ ENDWHILE LDR R0, #0, R1 LDR RO, #2, R2 BL PRINT_R1_R2 LDR R0, #4, R0 BRA WHILE ENDWHILE
^it's a linked list, next address into r0, occupies 3 words per element (2data, 1&next) or 9 words per element
MSBit in bit 2, LSBit bit 5. Use (m/lsb):inst&0b0010. where 1 is the wanted bit for each then (msb<<1)|lsb.
Stack Inside CPU: +ve=fast access, fewer clock cycles, low latency. -ve=small stack, expensive, overflow no long calls etc.
COMP inst: its useless because XOR does the same thing (dst^=src(0xFF))
Zero-address arithmetic: in the stack, +ve: no operand=simpler format, less instruction fetches (faster for simpler operations since less instruction fetches.-ve: more MEM access for push/pop, slower, less flexible. 1-2address: more flexible and can be optimized for performance w registers use.
Forward referencing loop: Post-test(do-while): eg.: LOOPST: CMP R0, #0; BNE LOOPST Bra if(r0!=0)
No-Operation Instruction: used for delays, debugging, doesn't do anything, can avoid hazards in pipeline.eg in xm:MOV r0,r0 create a delay.
Integer Function (Return Value to the Caller): MOV #5,R1; MOV #3,R2; CALL ADD_FUNC; MOV R0,Result
ADD_FUNC: MOV R0, RESULT.
ADD_FUNC: ADD R1, R2; MOV R2, R0; RETorBRA
SWITCH STATEMENTS=ARRAY?: both can map set of inputs to corresponding output values/actions. Eg: (switch case) with 0,1,2 output print0,1,2 and array of elements or function pointers like this: void f0(); void f1(); void f2();
main: void (*cases[3])()={f0,f1,f2};
if (condition) {call cases[i or value]}. this array will work identical to switch case as it maps input to output, control flow of code based on conditions, use relative addressing so they are fast as they jump through. Contiguous access to its indexed addressing.
Device Status Change: cpu continuously, periodically, actively checks the status register of the device to see if it needs attention, inefficient, introduce delays.
Device Interrupt: immediate interrupts caused, more efficient, no unnecessary checks, uses Interrupt service routine
Halt instruction: +ve: simple, minimal encoding. -ve: conflict with no-operation instruction, as they set 0x0 as the default empty instr. Slot. When its put into register and its opcode is 000 then the register can be seen as empty. Usually 0xffff is used for halt
--------—Program State:—————
has elements and is maintained in reg(current values), ram(below) and os(context switching, saving/restoring states).elements are: **PC**: holds address of next instruction to be executed.
Registers**: General-purpose registers:** Hold temporary data, intermediate results, and variables.
Special-purpose registers: Include the stack pointer (SP), base pointer (BP), and status registers (flags), which manage specific functions like stack operations and condition flags (e.g., zero, carry, overflow). **RAM:** Stack: Used for function calls, local variables, and return addresses. The stack grows and shrinks as functions are called and return. Heap: Used for dynamic memory allocation. Managed by the program and can grow in size as needed. Global/Static Data: Contains global variables and static local variables that persist for the program's duration. **PSW:** zero flag, carry flag, sign flag, overflow.
Open File Descriptors and I/O State
The state includes information about open files, network connections, and other I/O devices the program interacts with.
Execution Stack
The stack frame for the currently executing function includes the return address, parameters, local variables, and saved registers.
Cache organization: direct-Mapped Cache: each block of main mem is one cache line, simple and fast. **Fully Associative Cache:** any block can be loaded into any line of cache, more flexible+ less conflicts
PDP11 question: Register (**mode 000**): The operand is the register. No additional bytes needed.
Immediate (**mode 010**): A 16-bit data value follows the instruction. Requires 2 additional bytes.
Absolute (**mode 011**): A 16-bit address follows the instruction. Requires 2 additional bytes. Index (**mode 110**): A 16-bit offset follows the instruction. Requires 2 additional bytes.
Minimum size=2 bytes (instruction) +0 bytes (SRC)+0 bytes (DST)=2 bytes
Maximum Number: of Bytes
Maximum size=2 bytes (instruction)+ 2 bytes (SRC)+2 bytes (DST)=6 bytes
ORG #28
Main: MOVLZ LOOP, R3;
LD R3, R0;
LOOP: ST R0, -R3;
BR LOOP; DONE: END Main
^NO it never reaches DONE, continuously executes st line.
STACK:
typical Location: In most systems, the stack is located at the higher end of the memory address space and grows downwards towards lower memory addresses. This arrangement allows dynamic memory allocation (heap) to grow upwards from the lower end of the memory address space, maximizing the use of available memory.
Purpose:1-Function Call Management:
a.Local Variables: Temporarily store variables that are local to functions.
b.Return Addresses: Store the return address so that the CPU knows where to continue execution after a function call.
c.Function Parameters: Pass arguments to functions.
2.Control Flow:
aSaving Context: Save the state of the CPU (registers, program counter) during interrupts or context switches.
bNested Function Calls: Manage nested function calls and recursive functions by maintaining the correct return addresses and local variable states.
Implementing PUSH and PULL without Dedicated Instructions:
MOV R0, -(SP) ; (PUSH)Decrement the stack pointer and store the value from R0 to the stack. (PULL): MOV (R5)+, R0 ; Load the value from the stack into R0 and increment R5
Passing 3 16bit fields by value Question:
Sample:
.WORD 0x0000 ; f1 .WORD 0x0000 ; f2 .WORD 0x0000 ; f3
// asm to load f1 and f2 to r4 and r5
Call Subr1
St in f3
Subr1: ADD R4, R5 ; Add f1 and f2 (R4 + R5) MOV R5, R4 ; Store the result back in R4 (f3) BRA
Passing 3 16bit fields by refrence Question:
MOV #0x1000, R1
CALL Subr2
Subr2: LD (R1), R2 ; Load f1 into R2 LD 2(R1), R3 ; Load f2 into R3 ADD R2, R3 ; Add f1 and f2 (R2 + R3) ST R3, 4(R1) ; Store the result in f3 BRA
ADDING LDI&STI:
New instructions in ISA(opcodes&operand options), emulator should include decoding, +ve to our system is simplicity and

readability in code, efficiency(easier to make arrays), less instruction count.

BASEPOINTER:
or Frame Pointer (FP), is used to reference the base of the current stack frame in functions. It helps access function parameters and local variables within the stack frame.
 It remains constant throughout the function execution, unlike the Stack Pointer (SP) which changes as values are pushed and popped.

STACKPOINTER:
points to the top of the stack, which is the last used memory address in the stack. is used to manage the stack for function calls, returns, local variable storage, and temporary data. gets incremented or decremented as data is pushed to or popped from the stack.

LINKREGISTER: Link Register (LR) stores the return address when a function call is made saves next PC and goes back to it after function is done.(BL).

Data hazard: when the data(pc) is not ready for the next instruction requiring it

Control hazard: when an instruction (or instructions) following a branch execute when they should not.

```
; Structure example
; ECED 3403
; 25 Jun 2024
;
; struct stex
; {
;           unsigned short V0;
;           char V1;
;           short V2;
; };
;
; struct stex S1, S2;
;
; Define offsets
;
V0          equ         $0
V1          equ         $2
V2          equ         $4
;
; Data - reserve space for
structures
;
            DATA
            org         #100
S1          bss         $6          ;
6 bytes
            bss         $2          ;
Fill
S2          bss         $6          ;
6 bytes
;
; Code - initialize S1
;
            CODE
            org         #1000
;
StrtEx      movlz       S1,R0
            movh        S1,R0
;
; Initialize S1
;
            movls       #FF,R1
            str         R1,R0,V0
;
            movlz       'A',R1
            str.b       R1,R0,V1
;
            movlz       #EE,R1
            str         R1,R0,V2
;
; Initialize S2 from S1
;
            mov         R0,R2
            add         #8,R2       ;
Addr of S2
;
; s2.v0 <- s1.v0 + 2
;
            ldr         R0,V0,R1
            add         #2,R1
            str         R1,R2,V0
;
; S2.v1 <- S1.v1 + 1
;
            ldr.b       R0,V1,R1
            add         #1,R1
            str.b       R1,R2,V1
;
; S2.v2 <- S1.v2 - 2
;
            ldr         R0,V2,R1
            sub         #2,R1
            str         R1,R2,V2
;
Done
            end         StrtEx
```

SIZE OF VARIABLES:

Char	1 byte	%c, %s
Short	2 bytes	
Int	4 bytes	%d
Long	4 – 8 bytes	%l
Float	4 bytes	%f
Double	8 bytes	%lf

WAYS TO PRINT:

Hex	%x
Unsigned int	%u
Octal	%o

Note: 1 byte = 8 bits; 1 nibble = 4 bits

CISC vs. RISC machines
CISC-Complex Instruction Set Computer:
-Most instructions can generate EA:
ADD mem1,mem2
SUB #4,mem1
-Combinations of registers and memory permitted:
ADD R1,mem2
SUB R1,R2

• RISC – Reduced Instruction Set Computer (ours):
Limited number of instructions can generate EA:
LOAD mem1,Rx
STORE Rx,mem2

CODES THAT SET PSW:

```
Z: ; Clear R0 and R1
MOV #0, R0      ; R0 = 0
MOV #1, R1      ; R1 = 1

; Subtract R1 from R1
SUB R1, R1      ; R1 = R1 - R1
(result is 0)
; Z flag is set because the result
is zero
C:
; Load values into R0 and R1
MOV #0xFFFF, R0   ; R0 = 0xFFFF
(65535 in decimal)
MOV #1, R1      ; R1 = 1

; Add R0 and R1
ADD R1, R0      ; R0 = R0 + R1
(0xFFFF + 1 = 0x10000)
            ; C flag is set because
there is a carry out (0x10000
cannot be represented in 16 bits)
V:
; Load values into R0 and R1
MOV #0x7FFF, R0   ; R0 = 0x7FFF
(32767 in decimal, max positive
16-bit signed integer)
MOV #1, R1      ; R1 = 1

; Add R0 and R1
ADD R1, R0      ; R0 = R0 + R1
(0x7FFF + 1 = 0x8000)
            ; V flag is set because
the result (0x8000) is negative in
signed 16-bit representation
N:
; Load values into R0 and R1
MOV #0x0001, R0   ; R0 = 1
MOV #0xFFFF, R1   ; R1 = 0xFFFF
(-1 in signed 16-bit
representation)

; Subtract R0 from R1
SUB R0, R1      ; R1 = R1 - R0
(0xFFFF - 1 = 0xFFFE)
            ; N flag is set because
the result (0xFFFE) is negative in
signed 16-bit representation
```

Mnemonic	Instruction	PSW bits to inspect			
		V	N	Z	C
BL	Branch with Link	-	-	-	-
BEQ/BZ	Branch if equal or zero	-	-	= 1	-
BNE/BNZ	Branch if not equal or not zero	-	-	= 0	-
BC/BHS	Branch if carry/higher or same (unsigned)	-	-	-	= 1
BNC/BLO	Branch if no carry/lower (unsigned)	-	-	-	= 0
BN	Branch if negative	-	= 1	-	-
BGE	Branch if greater or equal (signed)	N ⊕ !V		-	-
BLT	Branch if less (signed)	N ⊕ V		-	-
BRA	Branch Always	-	-	-	-

Branching with Link and Branch Always are unconditional. When executed, control passes to the address of the label. The remaining branch instructions are conditional, meaning that one or more of the PSW bits must be inspected.

Branch with Link is a subroutine call. The return address is saved in the Link Register before control passes to the label. The subroutine can return to the caller by storing LR in the PC.