2024/05/04(05/20 수정)

컴퓨터의 개념 및 LC3 ISA

Brief History of Computing

-1940's : special-purpose computers, 배선 바꿔가며 프로그램
-Early 1950's : general-purpose computers, 한번에 하나의 프로그램만, cpu fetch-execution cycle(프로그램 저장 시작)
-Mid 1950's : batch programming, 여러 프로그램 등록시 순차 처리, resident monitor(first primitive version of system software)
-Early 1960's : Multiprogramming, 한번에 여러 프로그램 올리고 하나 idle이면 다른거 실행
-Mid 1960's : Timesharing, 주기적으로 task바꿈

Moore's Law / Amdahl's Law
무어의 법칙 : 24개월마다 트렌지스터 집적도가 2배씩 향상

Amdahl's Law : $\frac{1}{1-P+\frac{P}{n}} = speedup$

성능 증가의 3가지 제한 요인
-ILP Wall : 병렬화 한계
-Memory Wall : 메모리 느림
-Power Wall : 전력소모↓ 힘듦
*Power Wall->multicore로 보완
-> 현대에는 특화된 명령어, 프로세서

Big Idea of Computing
-Abstraction : CS는 추상화의 반복
-Universal Computing Device : 모든 계산 가능한 것은 어떤 컴퓨터든 계산 가능(모든 컴퓨터는 universal turing machine)
Bits and Data Repersentation

transistor : high or low volatage상태
-0(low)/1(high)로 데이터 표현 : Bit

의미있는 정보가 되기 위해선 문맥상에서 bit들이 해석될 필요가 있음
Information = Bits + Context
*1byte = 8bit, 1nibble = 4bit, 0x : 16진법, 0 : 8진법, 0b : 2진법(LC3는 b만)

Bit 연산
-and, or, not, xor
-bit shifht(logical/arithmetic)
-logical and, or, not(early terminaton)

Two's Complement Representation
-음수를 표현할 때는 절댓값의 2의 보수 이용해 표현(젤 위의 bit -라 생각과 동일)
-unsigned와 동일하게 더하기, 곱하기 가능(곱하기는 위쪽은 다름, 날라감)
- -x = ~x + 1 -> -x + x = 0

Integer Expansion and Truncating
-expansion (길이 늘리기) : signed / unsigned 존재

2024/05/04(05/20 수정)
-truncating(길이 줄이기)

Floating Point Representation(IEEE 754)

S / exp(E) / frac(M)

E 값에 따라 3가지 의미로 바뀜
-normalized values(대부분의 경우)

$$V = (-1)^S \times M \times 2^{E-bias}, M = 1.frac$$

-denormalized values(E = 0...0)

$$V = (-1)^S \times M \times 2^{1-bias}, M = 0.frac$$

0 완전하게 표현 가능 & 작은 숫자 좀 더 세밀하게 표현 가능
-special values(E = 1....1)
frac이 0이면 ±INF 아니면 NaN

Von Neumann Model
폰 노이만이 제안한 컴퓨터 구조, memory에 프로그램 저장하고 fetch-execution cycle따라 실행

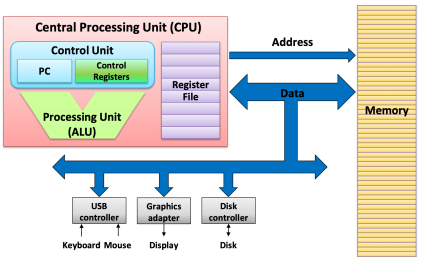

그림 1 Von Nenumann Model

-Memory : usually volatile(전기 끄면 사라짐), 보통 byte-addressible(byte단위로 접근 가능, LC3는 word addressible)
- Read : MAR에 주소 저장 -> write signal 0 -> MDR 읽음
- Write : MAR에 주소 저장 -> write signal 1 -> MDR 적음
-CPU : CU / ALU 기본 구성요소
* PC/IR도 register, ALU는 com circuit이고 보통 fixed sized word

Computer Circuit
0/1표현 -> Logic Gate -> Comb. Circuit / Seq. Circuit
-Logic Gate : input 받아서 output 줌. 연속적으로 변화, delay 약간 있음
-Comb. Circuit : Logic Gate들의 조합, Output이 input에만 의존
: BitEquity, Bit-Level Multiplexor, Word Equity, Word-Level Multiplexor
-Seq. Circuit : output이 input과 전 상태에 의존, clock signal에 따라 변화(보통 rising edge에 실행)
: Register, Accumulation Circuit, RAM

-CPU에서 data처리 과정 만드는 것 : building a datapath 라고 함(one clock cycle에 datapath 따라 한번 실행)

CPU Instruction Cycle
Fetch->Decode->Execude->Fetch2->...
크게 보아서 보통 이런 cycle을 따름. 한 사이클이 한 clock 동안에 일어나는 것은 아님. 한 clock에는 이것의 일부분이 일어남.

LC3의 Instruction Cycle
Fetch -> Decode -> Evaluate Address -> Fetch Operands -> Execute -> Store

2024/05/04(05/20 수정)

Fetch : MAR<-(PC)
PC = (PC) + 1 ;이 위치임이 중요
MDR <- Mem[MAR]
IR <- MDR

ISA 기본 개념

Hardware과 Software의 경계. 프로그래머에게 필요한 부분만 추상화한 것. RISC/CISC로 구분.

ISA의 구성요소

-Type, Size of Operands
-Operations ; Arithmetic / Data, Logic / Control의 3 종류로 구분
-Memory Addressing : pc related mode(주소 = (PC) + PCoffset값), base-offset mode(주소 = SR + offset값), indirect mode(주소 = 메모리[(PC) + PCoffset]), indeirect는 오래걸려서 보통 지원 안함
-Instruction Encoding
*addressing mode : immediate, register, memory

LC-3 Technical Info

-Memory : 2^16개의 16-bit wide word
-Register : 8개의 general purpose register, R6 : stack pointer, R7 : link
*LC-3는 16bit data type만 지원(signed int, unsigned int, instruction encoding)
*condition code존재-게산 결과 negative, zero, positive여부에 각각 하나씩 할당

ISA of LC-3

*opcode와 mnemonic 개념 알기

15 14 13 12 ... 5 0

| opcode | ...

Arithmetic & Logic Operation

AND / NOT/ ADD의 3개 지원
모두 condition code변경
-AND
AND DR, SR1, SR2 ; DR = SR1 & SR2
AND DR, SR1, #5 ; DR = SR1 & 5
-ADD
ADD DR, SR1, SR2 ; DR = SR1 + SR2
ADD DR, SR1, #5 ; DR = SR1 + 5
-NOT
NOT DR, SR ; DR = ~DR
NOT은 immediate value(ISA에서 바로 꺼내 쓸 수 있는 값) 지원 안함, 또한 두 값 모두 immediate인거는 다 지원 안함(코드로 안하고 계산해서 넣으면 되기 때문)

Data trasfer Operation

LD, LDR, LDI, ST, STR, STI, LEA지원
LEA제외 모두 condition code 바꿈
-LD
LD DR, #offset9
LD DR, label ; label로 한 경우 컴파일러에서 자동으로 offset으로 바꿔줌

2024/05/04(05/20 수정)
-LDR
LDR DR, R1, #offset5
-LDI
LDI DR, #offset9
-ST류 : LD와 동일한 모양으로 씀. 이번에는 왼쪽에 있는 데이터를 오른쪽으로 옮김
-LEA : offset값 이용해 DR에 자동으로 절대 주소값 넣어줌
LEA DR, #offset9
LEA DR, label

Control Operation
BR, JMP, JSR, JSRR, RET, TRAP
-BR
BRp #5
BRnz label ; n or z 가 참일 때 label로 이동
-JMP
JMP SR ; SR로 jump
-JSR : jump하고 R7에 직전 주소 저장
JSR #7
JSR label
-RET : R7으로 돌아감
RET
-JSRR : JSR과 같은데 register인거만 차이
JSRR SR
-TRAP : user에게 권한이 없는 것 os가 실행해줌(대표적으로 I/O)
TRAP x23
x20 ; GETC, x21 : OUT, x22 : PUTS, x23 : IN, x24 : PUTSP, x25 : HALT

프로그램 설계하기
computer을 이용한 문제해결 = problem solving + debugging
programming은 problem solving
-sequential, iterative, conditional decomposition
각각의 구조에 대해서 알 필요가 있음
-sequential decomposition : 그냥 이어서
-conditional decomposition : BR이용해 하나는 jump하고 하나는 그러지 않음
-iterative decomposition : BR이용해 끝내는건지 판단->아니면 내부 실행, 판별할 때 쓰이는 것 업뎃->젤 위쪽으로 다시 돌아감

참고문헌
서울대학교 2024-1 컴퓨터의 개념 및 실습(Digital Computer Concepts and Practice) Lecture & Lecture Slide