CA02

- Arithmetic Operation

add a, b, c (a= b+ c)
sub a, b, c (a=b-c)

register 사용

in MIPS : 32×32bit , 0~31로 넘버링 , 32bit data A.K.A word.

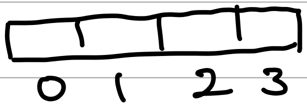
$ t0 ··· t9 temp value
$ s0 ··· s7 saved variable

모든계산은 레지스터로만! (register to register / · 다시오기)

메모리 $\xrightarrow[\text{Load}]{}$ 레지스터 → 계산 → 레지스터 $\xrightarrow[\text{Store}]{}$ 메모리
결과

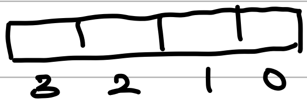
하나의 주소는 8 bit (1byte)
따라서 word 주소는 4의 배수 임.

- Big Endian (MIPS)



채우는 순서

Little Endian



- 예제

g = h + A[8]

lw $t0, 32($s3) $s3는 A의 base address

↓

8 × 4

- Immediate Operand

addi $s3, $s3, 4

-1] 음수 더하기

$zero 0번 레지스터 (덮어쓰기 안됨)

Signed 2's complement 사용

- Sign extention

- MIPS 32 ISA

R-format	op	rs	rt	rd	shamt	funct
	6	5	5	5	5	6

0	17	18	8	0	add
R-format	S1	S2	목적지		

- bitwise manipulation

- shift

sll logic 연산이어서 0으로 채움 (sla는 부호비트로 채움)

- logical

- not

a nor 0 = not a 임을 활용.

- I-format

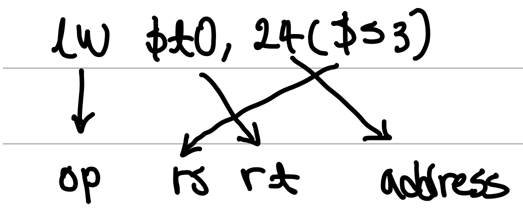
op	rs	rt	const or address
6	5	5	16

$-2^{15} \sim 2^{15}-1$ 까지만 표현가능

- memory access

lw $t0 4($s3)   sw $t0, 8($s3)   (word)

$s3를 base address register + 4, 8 등 offset을 더한 주소


lw $t0, 24($s3)

op rs rt address

24+$s3 인 주소의 데이터를 load

lb sb 명령어 (byte) 오른쪽 끝에 load, store

- Conditional operation

branch

beq rs rt L1 rs==rt이면 L1 instruction 실행

bne rs rt L1 rs!=rt이면

jump

j L1 그냥 L1으로 jump

PC+4로 계속 업데이트됨.

넘어갈 instruction 주소는 현재 명령에서 Program Counter로 상대적으로 찾는다.

$-2^{15} \sim 2^{15}-1$ 까지의 명령으로 이동가능

- Set on less than

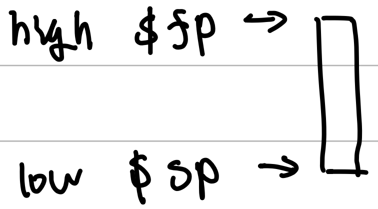
slt $t0, $s0, $s1 $s0 < $s1 : $t0=1

else : $t0=0

blt	less than	bgt	greater than
ble	less than or equal	bge	greater than or equal

- jal   jump and link : 함수 실행 (함수 끝나고 돌아올 PC+4 $ra에 저장후 함수로 jump)
  jr   return: $ra 를 PC 로 복사.

- stack



- atomic exchange support

                                        load
  load linked   ll rt, offset(rs)
  store conditional   sc rt, offset(rs)
                               ↓
                               1 이면 atomic
                               0 이면 바뀜

- pseudo instruction
  move → add $t0 $zero $t1
  blt → slt + bne

- powerful instruction → higher performance? 구현 어려움
  write assembly code → high performance? 에러↑ 생산성↓
  backward compatibility → instruction set 변동x ? 더 추가했다.

주소는 4씩 증가!