1. Lecture 2 and lab 2 notes

Instruction categories

1. load and store
2. computation
3. jump and branch
4. floating point number

32*32-bit register

- 32 × 32-bit registers

Name	Register number	Usage
$zero	0	The constant value 0
$v0–$v1	2–3	Values for results and expression evaluation
$a0–$a3	4–7	Arguments
$t0–$t7	8–15	Temporaries
$s0–$s7	16–23	Saved
$t8–$t9	24–25	More temporaries
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

we can use to store values: **$t0 - $t9, $S0 - $S7**

不过从习惯上，$t0 - $t9 存放暂时值（temporary, $s0 - $s7存放初值和末值（saved values）

Arithmetic Instructions

1、addition

1. register adding

```
add rd, rs, rt; //rd = rs + rt
sun rd, rs, rt; //rd = rs - rt
```

2. adding immediate

```
# addi rd, rs, 20表示将rs和20加起来，放在rd之中
addi $t0, $t0, 1;
# no substraction, just use a negtive number instead
```

3. constant zero

```
# equivalence
addi $t2, $t1, 0;
add $t2, $t1, $zero;
# zero cannot be overwritten
```

2、loading and storing

1. loading
    1. format

```
# load a word
lw $t0, 32($s3); # 将s3偏移32bytes后，取一个word(4bytes)，放在t0寄存器中(注意不是在赋值地址)

# load a byte
lb $t0, 32($s3); # 将s3偏移32bytes后，取一个bytes，放在t0寄存器(32bits)中(注意不是在赋值地址)

# load a half-word
lh $t0, 32($s3); # 将s3偏移32bytes后，取一个half-word(2 bytes)，放在t0寄存器(32bits)中(注意不是在赋值地址)


```

    2. sign extension

    `lb` 和 `lh` 指令用于从存储器中读取字节和半字数据，并将其符号扩展为 32 位。符号扩展的目的是将原始数据的符号位扩展到更高位，以保证符号位的正确性。例如，如果读取的字节或半字数据的最高位为 1，表示它是一个负数，则符号扩展会将这个最高位复制到更高的位上，以保持数据的符号性。如果读取的字节或半字数据的最高位为 0，则符号扩展会将更高的位全部填充为 0，以保持数据的正确性。(by chatgpt)

    3. load an unsigned number(without sign extension)

```
# load a unsigned word
lwu $t0, 32($s3); # 将s3偏移32bytes后，取一个word(4bytes)，放在t0寄存器中(注意不是在赋值地址)

# load a unsigned byte
lbu $t0, 32($s3); # 将s3偏移32bytes后，取一个bytes，放在t0寄存器(32bits)中(注意不是在赋值地址)

# load a unsigned half-word
lhu $t0, 32($s3); # 将s3偏移32bytes后，取一个half-word(2 bytes)，放在t0寄存器(32bits)中(注意不是在赋值地址)
```

2. storing

```
# store a word
sw $t0 32($s3); # 将t0的第一个word存进$s3+32bits

# store a byte
sb $t0 32($s3); # 将t0的第一个字节存进$s3+32bits位置处

# store a half-word
sh $t0 32($s3); # 将t0的第一个half-word存进$s3+32bits位置处
```

3. shifting

```
# sll: 移位后，空出的位都填充为 0。
# 移位操作可能会导致溢出，左移的位数超过了数据类型的位数，那么操作结果可能是不确定的。
sll $t0, $s1, 4; # reg t0 = s1 << 4 bits

# srl: 移位后，将最高位的空出的位都填充为 0。
srl $t0, $s1, 4; # reg t0 = s1 >> 4 bits

# sra: 移位后，将最高位（即符号位）复制到空出的位上，这样可以保持操作数的符号不变。
sra $t0, $s1, 4; # reg t0 = s1 >> 4 bits

# 没有sla
```

4. logic

```
# bitwise operation
and $t0, $t1, $t2; # t0 = t1 & t2
or $t0, $t1, $t2; # t0 = t1 | t2
nor $t0, $t1, $t2; # t0 = ^(t1 | t2)
xor $to, $t1, $t2;
/*没有not, not等价于
    nor $t0, $t1, $zero;*/
```

5. conditonal operation

1. conditional branch

```
beq rs, rt, L1; # if rs == rt, jump to label L1
bne rs, rt, L1; # if rs != rt, jump to label L1
```

2. Unconditional branch

```
j L1; # jump to L1
```

some data type in mips

```
.data
    s1: .ascii "welcome " # 没有\0的字符串
    sid: .space 9 # 一串空格，长度为9bytes
    e1: .asciiz "to Mips world" # 自动补\0的字符串
    c1: .byte 'A' # 一个字符（本质上就是一个byte）
    i1: .word 32 # 存放一个整数
```

instructions and applications

1. data flow between registers and memory

```
# b = a + 1
# from memmory to register: loading
lw $t0, a

addi $t1, $t0, 1

# from register to memmory: storing
sw $t1, b
```

2. address assignment

```
# with address assignment, we can modify the memory directy
# assign a address directly
la $t0, 0x100000000

# assign an address from a data varible
la $t0, a
# we can modify a, a[1]... now
```

(by chatgpt)

3. syscall:
    1. dependency: we can use syscall to call system instruction, with $v0 to determine which system instruction to be call, and $a0, $a1, $a2, $a3, $f11, $f12, to determine the arguments
    2. example

```
li $v0, 8 # read a string
la $a0, sid # store address of memory to $a0
li $a1, 9 #set the max length of reading
syscall # call system instruction
```

4. set a boolean: slt, sltu

```
# set less than
slt $t1, $t2, $t3 # t1 = (t2 < t3)

# set less than by unsigned comparision
sltu $t1, $t2, $t3 # t1 = ((unsigned)t2 < (unsigned)t3)
```