СӘТБАЕВ УНИВЕРСИТЕТІ

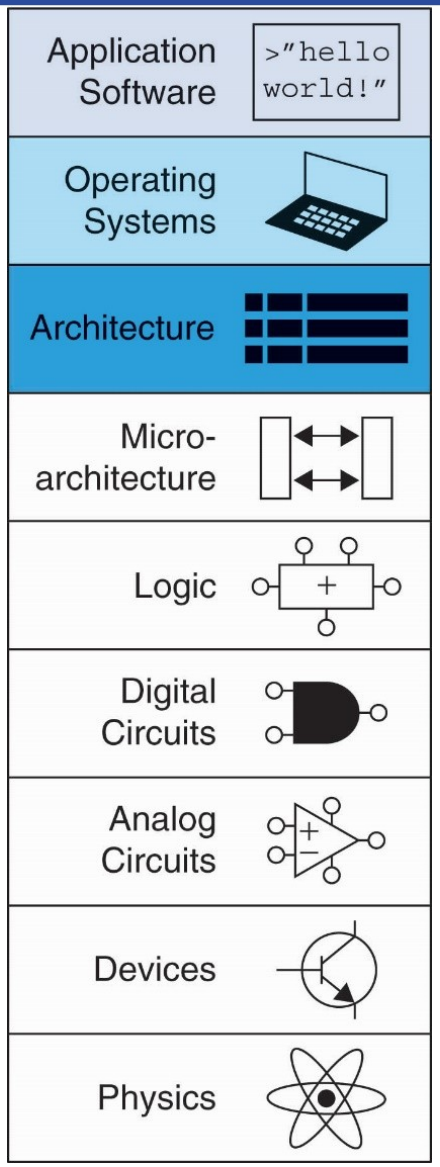
SATBAYEV UNIVERSITY

Компьютер архитектурасында бағдарламалық қамтамасыз ету мен аппараттық құралдардың өзара әрекеттесу деңгейлері

Оқытушы: Жекамбаева М.Н.
«Программалық инженерия»
кафедрасының профессоры
m.zhekambayeva@satbayev.university

Қарастыратын негізгі сұрақтар

- **Ассемблер тілі**
- **Машиналық тіл**
- **программалау**
- **Адрессациялау режимі**
- **Ассемблерлеу және жүктеу**
- **Қосымша ақпарат**



Ассемблер тілі

- **Инструкция:** процессор қолданатын команда
  - **Ассемблер тілі:** адам оқи алатын процессор нұсқауларының командасы (нұсқаулар мен олардың операндалары мәтін түрінде берілген)
  - **Машиналық тіл:** процессор үшін түсінікті нұсқаулардың форматы (екілік сандар 0 мен 1)

MIPS архитектурасы:

- 1980 жылдары Джон Хеннесси мен Стэнфордтағы әріптестері әзірледі.
- Silicon Graphics, Nintendo және Cisco өнімдерін қоса көптеген коммерциялық жобаларда қолданылады

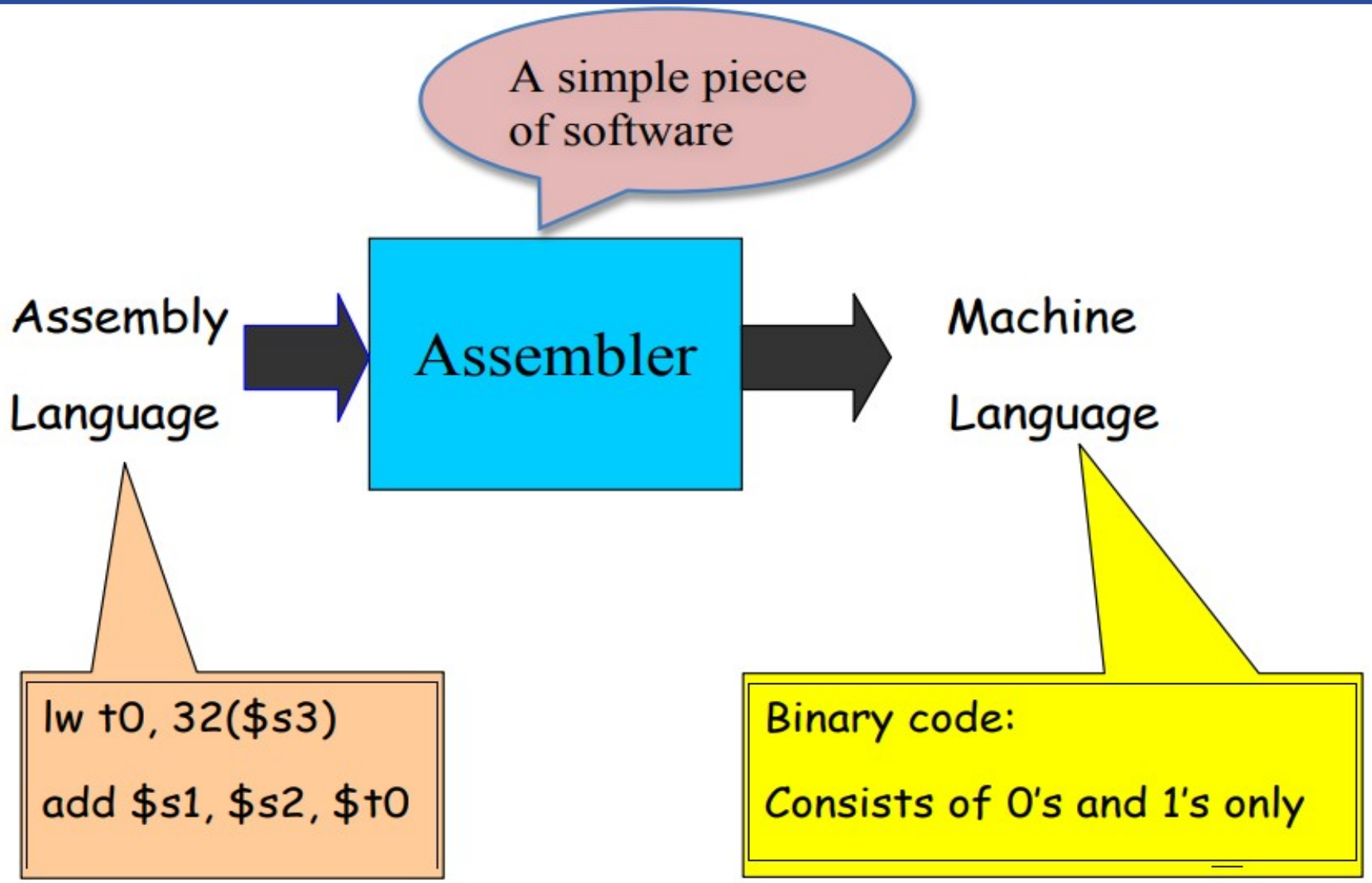
Джон Хеннесси

- Стэнфорд университетінің президенті (ректоры)
- 1977 жылдан Стэнфордта информатика және инженерия профессоры
- Дэвид Паттерсонмен бірге олар қысқартылған нұсқаулықтар жиынтығымен RISC есептеу жүйелерін қолдануды ұсынды (қазіргі кезде көптеген мобильді және ендірілген жүйелер RISC процессорларына негізделген)
- 1984 жылы Стэнфордта MIPS архитектурасы әзірленді және MIPS Computer Systems негізін қалады
- 2004 жылы 300 миллионнан астам MIPS процессорлары сатылды

Құрылымдық архитектурасының принциптері

Хеннесси мен Паттерсон құрылымдық архитектурасы негізгі принциптерін тұжырымдады:

- біртектілік - қарапайымдылықтың кілті
- әдеттегі сценарий жылдам болуы керек
- кішкене болу - тез бол
- жақсы дизайн жақсы келісімді қажет етеді


A simple piece of software
Assembly Language
Assembler
Machine Language
lw t0, 32($s3)
add $s1, $s2, $t0
Binary code:
Consists of 0's and 1's only

1- архитектуралық құрастыру принципі

Бірыңғайлылық - қарапайымдылықтың кілті

Инструкциялардың бірыңғайлылық форматы жақсы құрастыру принципі

- Операндар санының ұқсас болуы (екі негізгі операндалар, бір жиынтық операнд)
- Бұл тәсіл аппараттық құралдарды енгізуді айтарлықтай жеңілдетуге мүмкіндік береді - сол форматтағы нұсқауларды декодтау және орындау оңайырақ болады

Операнды: Регистры

- MIPS құрамында 32 биттік 32 жалпы регистр бар
- Регистрлер жадқа қарағанда жылдамырақ (олар кіруге аз уақыт алады)
- MIPS-те арифметикалық және логикалық нұсқаулардың регистрлері немесе тұрақтылары (өнімділікті жақсарту және аппараттық құралдарды енгізуді жеңілдету үшін)
- MIPS «32 биттік архитектура» деп аталады, себебі ол 32 биттік деректермен жұмыс істейді және 32 биттік нұсқаулықтарға ие.

Қосындысын алу: нұсқаулығы (инструкция)

C дағы жазылу

```
a = b + c;

a = b - c;
a = b / c;
a = b * c;
```

MIPS ассемблер коды

```
add a, b, c
sub a, b, c
div a, b, c
mul a, b, c
```

- **add:** мнемоника инструкцисы, қандай амалды орындауды көрсетеді
- **b, c:** мнемоникамен көрсетілген операция орындалатын бастапқы операндтар
- **a:** тағайындалған операнд, тағайындалған операнд (оған операцияның нәтижесі жазылған)

Азайту: нұсқаулығына мысал

- формат қосынды нұсқауларына өте ұқсас, тек sub азайту амалы мнемикалық коды бойынша ерекшеленеді

C дағы код

```
a = b - c;
```

MIPS ассемблердегі коды

```
sub a, b, c
```

- sub: мнемоника
- b, c: бастапқы операндалар
- a: тағайындалған операнд,

Қарапайым нұсқаулардың көп болуы

- Күрделі жоғары деңгейлі код бірдей форматтағы бірнеше қарапайым MIPS нұсқауларына түрлендіріледі

C дағы коды

```
a = b + c - d;
```

MIPS ассемблердегі код

```
add t, b, c  # t = b + c
sub a, t, d  # a = t - d
```

MIPS регистрлері жинағы

- $zero ($0) — 0 мәнін қайтарады, тек оқуға арналған .
- $at ($1) — процессор үшін уақытша регистр.
- $v0 — $v1 ($2 — $3) — функциялар қайтарған нәтижелер үшін.
- $a0 — $a3 ($4 — $7) — функция аргументтері үшін.
- $t0 — $t9 ($8 — $15,$24 — $25) — уақытша мәндерді сақтау үшін қолданылады
- $s0 — $s8 ($16 — $23,$30) — Тұрақта мәндерді сақтау үшін қолданылады.
- $k0 — $k1 ($26 — $27) — ОЖ ядросы үшін тіркелген.
- $gp ($28) — глобальды айнымалыларға көрсеткіші.
- $sp ($29) — стек көрсеткіш, стека жоғарғы адрес мәніне тең.
- $fp ($30) — фрейм көрсеткіш.
- $ra ($31) — функция шақырылған инструкция адресін қамтиды.
- $f0 – функцияларға қайтатын жылжымалы нүктелі мәндер үшін.
- $f4, $f6, $f8, $f10, $f16, $f18 – уақытша жылжымалы нүктелік деректер үшін.
- $f12, $f14 – өзгермелі нүкте функциясының параметрлері үшін.
- hi,lo — көбейту және бөлу кезінде қолданылады.
- pc — программна снағышы.

syscall жүйелік шақыру нұсқаулықтары

Оны қолданар алдында 1-ден 16-ға дейінгі қызмет кодтарының бірін $ v0 регистріне жүктеу қажет. операциялық жүйенің кодтары мен тиісті әрекеттер тізімі:

1 —$a0 регистрында орналасқан натуралды сандарды экранға шығару.
2 —$f12 регистрында орналасқан жылжымалы нүктелі санды экранға шығару.
3 — $f12 регистрында орналасқан жылжымалы нүктелі санды экранға шығару(екі есе дәлдікпен).
4 —$a0 регистрында орналасқан (нөлдік символ) жолды эканға шығару.
5 —$v0 натурал санды санап, оны тізілімге енгізу.
6 — жылжымалы нүктелі санды оқып, оны $f0 регистрге енгізу..
7 —жылжымалы нүктелі санды(екі есе дәлдікпен) санап $f0 регстрына жіберу.
8 — Жолды оқу, $a0 — жолдық буфер адресі , $a1 — неше символ оқу қажет.
9 — ?
10 — программадан шығу
11 — $a0 регистрынан символдарды шығарып алу.
12 — символдарды санап $a0 жіберу.
13 — файлды ашу, $a0 — адрес нөлдік-жол, файлдың аты беріледі, $a1 — флаг, $a2 — режим. $v0 операциясынан кейін файл дескрипторын қамтуы керек.
14 — файлдан оқу, $a0 — дескриптор, $a1 —кіріс буфер адресі, $a2 — қанша символ оқылды. $ v0 Операциядан кейін оқылған таңбалардың санын көрсетіледі.
15 — файлға жазу, $a0 — дескриптор, $a1 — шығыс буферінің адресі, $a3 — қанша символ жазу керек. $v0 операциядан кейін жазылған символдардың санын қамтиды.
16 — файлды жабу, дескриптор a0 де орналасқан.
nop — машиналық код 00000000 , ещқандай асер етпейді.
break — отладка үшін қолданылады

Тікелей немесе жанама адрестелу

- la — $x, var — RAM нан var жады адресін (.data да жарияланған) x регистрына көшіру.
- lw $x, s($y) —RAM да $x адресі жадынан сөзді, $y адресіне жүктеу. Адрестен регистрге (s) жылжытып толтыруға болады.
- li — $x, i16 — 16 немесе 32-биттік санды $x регистіріне жазу.
- sw — $x, s($y) — зарезервировать машинное слово, содержащее в регистре $x, в RAM памяти по адресу. Можно дописать смещение (s). $xрегистрындаға машиналық сөзді RAM жады бойынша резервтеп $y жазу. Тіркеліп жазуға болады

MIPS ассемблер маңызды нұсқаулықтары операциялық жүйе жүйелік шақыруларын қолданады.

2- архитектуралық құрастыру принципі

Әдеттегі сценарий жылдам болуы керек

MIPS архитектурасы тек қарапайым, жиі қолданылатын нұсқауларды қамтиды

Декодтау схемаларын аппараттық түрде енгізу және мұндай нұсқауларды орындау қарапайым, жылдам және чипте аз орын алады

Күрделі нұсқауларды (және сирек кездесетіндерді) компилятор бірнеше қарапайым нұсқауларға айналдырады

MIPS - қысқартылған нұсқаулықтар жиынтығы бар RISC процессоры - сол форматтағы қарапайым нұсқаулардың аз саны (RISC - Редукцияланған нұсқаулықтар компьютері)

Басқа архитектуралар, мысалы Intel x86, Complex Instruction Set Computers (CISC) мысалдары болып табылады.)

2- архитектуралық құрастыру принципі

Кішкене және жылдам болу

MIPS жалпы мақсаттағы регистрлердің аз саны қамтиды

- *регистрлер:*
  - ассемблер программасында $ кейін жазылады
  - Мысалға: $0, “нөлдік регистр”
- Add инструкциясын есімізге түсірсек

C дағы код

```
a = b + c
```

MIPS ассемблер коды

```
# $s0 = a, $s1 = b, $s2 = c
add $s0, $s1, $s2
```

Сөздерді адрестік жадттан оқу

- ***32-разрядты сөзді оқу үшін жадыдан жлпы қолданыс регистрі load қолданылады.***
- **Мнемоника:** *load word* (***lw***)
- **Формат:**

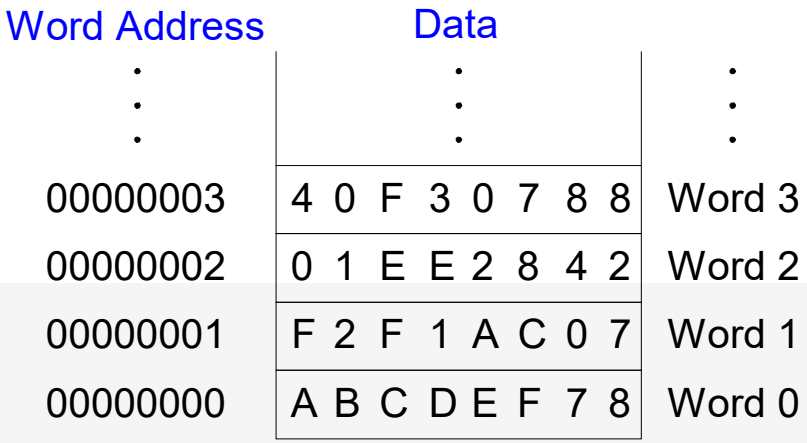
  lw $s0, 5($t1)
- **Расчет адреса слова в памяти:**
  - Регистрде сақталатын базалық адресті ($ t1) тұрақты ығысумен жинақтау қажет(5)
  - Адрес = ($t1 + 5)
  - Негізгі адресті сақтау үшін кез келген регистрді қолдануға болады.
  - **Результат:**
  - ($ T1 + 5) адресінде жадта сақталған мән $ s0 регистріне оқылады

Сөзді жадта сақтау адресі бойынша оқу

- **мысалы:** жадыдан сөзді оқып алу(адрестелу бойынш) $s3 регистрында 1-ші адрестен оқу
  - Адрес = ($0 + 1) = 1
  - Load инструкциясын орындағаннан кейін, $s3 регистрында 0xF2F1AC07 мәні болады

Ассемблердегі код

```
lw $s3, 1($0)  #1 адресі бойынша сөзді $s3-ке
   оқу(жүктеп алу)
```



айырмашылықтары
lw **r8,0(r4)**
және
mov r8, r4
Load word мәні "copy from memory", бірақ load word r4 регистрынан көшіреді жадттан емес

Жадқа адрестелген информацияны жазу

- Жалпы регистрдің мазмұнын жад ұяшығына жазу нұсқаулықтың көмегімен жүзеге асады *store*
- Мнемоника: *store word* (*sw*)
- Форматтың lw нұсқаулығымен көп ұқсастығы бар

Адрестлеу жадына сөзбе сөз жазу.

- Мысал: Мысал: $ t4 регистрінің мәнін 7 адрес бойынша жадқа (адрестік сөзбен) жазу (сақтау) қажет.
- жад ұяшығының мекенжайы негізгі адресті ($ 0) жылжыту (0x7) қосу арқылы алынады
- Адресін алу: ($0 + 0x7) = 7

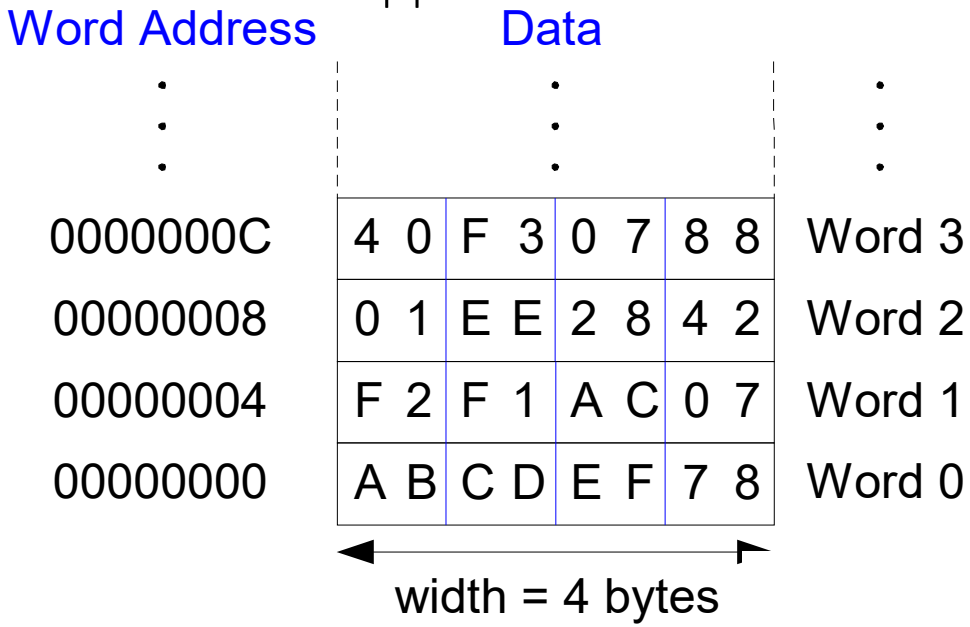
Есептеу ондық және он алтылық жүйеде де жазылуы мүмкін

Ассемблердегі код

```
sw $t4, 0x7($0)   # $t4 регистр мәнін сақтаймыз
                  # 7 адрестік жады ұяшығына
```

Жад адрестелетін байт

- Жадыдағы әрбір байт бірегей адреске ие
- Көршілес байт адрестері 1-ге ерекшеленеді
- Сөздерді де, жеке байттарды да оқи / жаза аласыз (load/store): байтты жүктеңіз (lb) және байтты сақтаңыз (sb)
- 32 биттік сөз = 4 байт, сондықтан әрбір келесі сөздің адресі 4-ке көбейеді

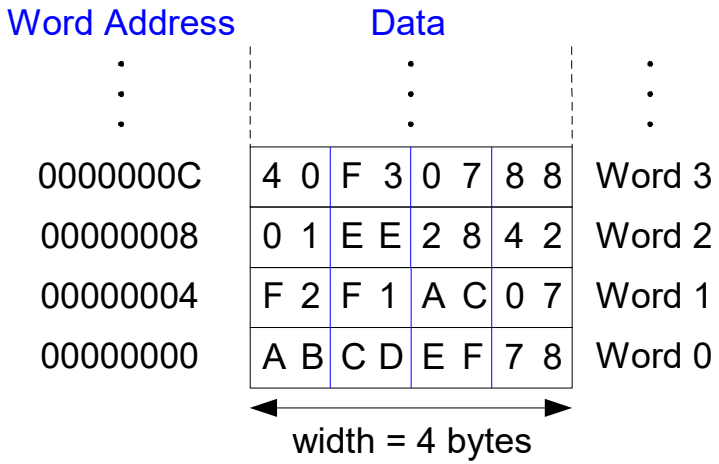


Жадта адрестелген байтты оқу

- Сөздегі адрестен байтқа өту үшін, сөздегі адресті 4 -ке көбейту керек(әр сөзде 4 байт бар). Мысалға:
- 2-ші сөз адресі байт бойынша жадыда: 2 × 4 = 8
  - 10-сөз адресі байт бойынша жадыда : 10 × 4 = 40 (0x28)
- **MIPS архитектурасында жады байттар бойынша адрестеледі, сөздермен емес**

Код ассемблер MIPS

```
lw $s3, 4($0)  #4-адрес мәнін $s3 регистрға жазу
```



Жадта адресіне сөзді жазу

- **мысалы:** $t7 регистрының 44 адресіне жазу керек болса (байт). Назар аударсаңыздар адрестік жылжыту ондық санау жүйесінде көрсетілген.

Код ассемблер MIPS

```
sw $t7, 44($0)  #$t7 мазмұнын 44 мекенжайы бойынша
   сақтаңыз
```

Машиналық тіл

- Инструкциялардың екілік санау жүйесінде көрсетілуі(0 және1)
- 32-биттік инструкция
  - Біртектілік - қарапайымдылықтың кілті : 32-бит деректер мен нұсқаулар

R-Типтік нұсқау

Assembly Code

```
add $s0, $s1, $s2

sub $t0, $t3, $t5
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Ассемблер кодындағы регистрлердің реті:

rd, rs, rt қосыңыз

I-Типті нұсқау

Assembly Code

```
addi $s0, $s1, 5
addi $t0, $s3, -12
lw   $t2, 32($0)
sw   $s1,  4($t1)
```

Field Values

op	rs	rt	imm
8	17	16	5
8	19	8	-12
35	0	10	32
43	9	17	4
6 bits	5 bits	5 bits	16 bits

Ассембле және машина тілдерінде регистрлердің әр түрлі орналасуына назар аударыңыз:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

Machine Code

op	rs	rt	imm	
001000	10001	10000	0000 0000 0000 0101	(0x22300005)
001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
6 bits	5 bits	5 bits	16 bits	

Көбейту, бөлу

- Нәтиже алу үшін арнайы регистрлар: `lo`, `hi`
- 32 × 32 көбейту, 64биттік резултат
  - `mult $s0, $s1`
  - Результат жазу керек {`hi`, `lo`}
- 32-биттік бөлу, 32-битті бүтін және қалдық
  - `div $s0, $s1`
  - бүтін `lo`
  - **Бөлудің қалған бөлігі** `hi`
- Арнайы регистрынан оқу `lo`/`hi` жалпы қолданыс регистрында
  - `mflo $s2`
  - `mfhi $s3`

Орын ауыстыру типтері:

- **Шартты**
  - Егер екі регистрдің мазмұны тең болса, берілген мекен -жайға өтіңіз(branch if equal, `beq`)
  - Егер екі регистрдің мазмұны тең болмаса, берілген мекен -жайға өтіңіз(branch if not equal, `bne`)
- **Шартсыз**
  - Константпен берілген адреске шартсыз өту(jump, `j`)
  - Тұрақты көрсетілген мекен -жайға шартсыз өту (jump register, `jr`)
  - 31 -регистрде процедурадан қайтару мекенжайын сақтай отырып, тұрақтымен көрсетілген процедураның мекен -жайына сөзсіз өту (jump and link, `jal`)

Басқарушы құрылым/ауысу

- j target — target меткасына өту.
- jr $x —x регистрінің мәні бар адреске өту.
- jal target — бағдарлама есептегішінің мазмұнын $ra регистріне көшіреді және мақсатты белгіге өтеді.

Шартты тармақталу

- b target —target меткаға нақты ауысу.
- beq $a, $b, target — target меткасына ауысу, егер a = b.
- blt $a, $b, target — target меткасына ауысу, егер a < b.
- ble $a, $b, target — target меткасына ауысу, егер a <= b.
- bgt $a, $b, target — target меткасына ауысу, егер a > b.
- bge $a, $b, target — target меткасына ауысу, егер a >= b.
- bne $a, $b, target — target меткасына ауысу, егер a <> b.

Шартты тармақталу beq

```
# Код ассемблере MIPS
 addi $s0, $0, 4           # $s0 = 0 + 4 = 4
 addi $s1, $0, 1           # $s1 = 0 + 1 = 1
 sll  $s1, $s1, 2          # $s1 = 1 << 2 = 4
 beq  $s0, $s1, target  # target ауысу
 addi $s1, $s1, 1       # инструкция орындалмайды
 sub  $s1, $s1, $s0     # орындалмайды
target:                    # метка
 add  $s1, $s1, $s0        # $s1 = 4 + 4 = 8
```

Условный переход, если не равно (bne)

```
# Код на ассемблере MIPS
  addi   $s0, $0, 4          # $s0 = 0 + 4 = 4
  addi   $s1, $0, 1          # $s1 = 0 + 1 = 1
  sll    $s1, $s1, 2         # $s1 = 1 << 2 = 4
  bne    $s0, $s1, target    # переход не выполняется
  addi   $s1, $s1, 1         # $s1 = 4 + 1 = 5
  sub    $s1, $s1, $s0       # $s1 = 5 - 4 = 1

target:
  add    $s1, $s1, $s0           # $s1 = 1 + 4 = 5
```

Безусловный переход (j)

```
# Код на ассемблере MIPS
  addi $s0, $0, 4                  # $s0 = 4
       addi $s1, $0, 1                   # $s1 = 1
       j      target                # переход на метку
target
       sra    $s1, $s1, 2       # не выполняется
       addi   $s1, $s1, 1       # не выполняется
       sub    $s1, $s1, $s0     # не выполняется

  target:
       add    $s1, $s1, $s0         # $s1 = 1 + 4 =
5
```

Высокоуровневые конструкции

- Оператор `if`
- Оператор `if/else`
- Цикл `while`
- Цикл `for`

Оператор if

Код на C

```
if (i == j)
  f = g + h;

f = f - i;
```

Код на ассемблере MIPS

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
    bne $s3, $s4, L1
    add $s0, $s1, $s2

L1: sub $s0, $s0, $s3
```

В данном случае код на C проверяет условие (`i == j`), а код на ассемблере проверяет противоположное условие (`i != j`). Результат один и тот же.

Оператор if/else

Код на C

```
if (i == j)
  f = g + h;
else
  f = f - i;
```

Код на ассемблере MIPS

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
      bne $s3, $s4, L1
      add $s0, $s1, $s2
      j   done
L1:   sub $s0, $s0, $s3
done:
      ... #следующие инструкции
```

Проверка неравенств

Код на C

```
// суммирует степени двойки
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
  sum = sum + i;
}
```

Код на ассемблере MIPS

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0, $t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```

$t1 = 1 если i < 101

Цикл while

Код на C

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

Код на ассемблере MIPS

```
# $s0 = pow, $s1 = x

        addi $s0, $0, 1
        add  $s1, $0, $0
        addi $t0, $0, 128
while:  beq  $s0, $t0, done
        sll  $s0, $s0, 1
        addi $s1, $s1, 1
        j    while
done:
```

Код на C проверяет условие (**`pow != 128`**), а код на ассемблере проверяет противоположное условие (**`pow == 128`**).

Результат один и тот же.