КАРХ: Тема_9: Машинен език. Видове инструкции

Въведение.

- **Асемблерният език** е за удобство на хората.
- Компютрите работят само с 1 и 0, за това мнемоничният код се транслира в представяне само с 1 и 0, което се нарича **машинен език**.
- В MIPS архитектурата се използват 32-bit инструкции:
  - Simplicity favors regularity: 32-bit data & instructions
- Има 3 основни формата на инструкциите:
  - **R-Type:** оперира с 3 регистъра;
  - **I-Type:** оперира с 2 регистъра и 16-bit константа (immediate);
  - **J-Type:** за преход – оперира с една 26-bit константа.

КАРХ: Тема_9: Машинен език. Видове инструкции

R-Type инструкции.

- Този тип инструкции използва 3 регистъра като операнди:
  - rs, rt: два регистъра източници (source registers)
  - rd: един регистър като получател на резултата (destination register)
- Инструкциите имат 6 полета – op, rs, rt, rd, shamt и funct.
- Значение на другите полета:
  - op: the *operation code* или *opcode* (0 за R-type инструкции).
  - funct: the *function* заедно с opcode, указва на процесора каква операция да извърши.
  - shamt: the *shift amount* – указание за shift инструкции, в другите случаи е 0. За всички R-Type инструкции shamt = 0.

R-Type

КАРХ: Тема_9: Машинен език. Видове инструкции

R-Type инструкции.

Пример:

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	

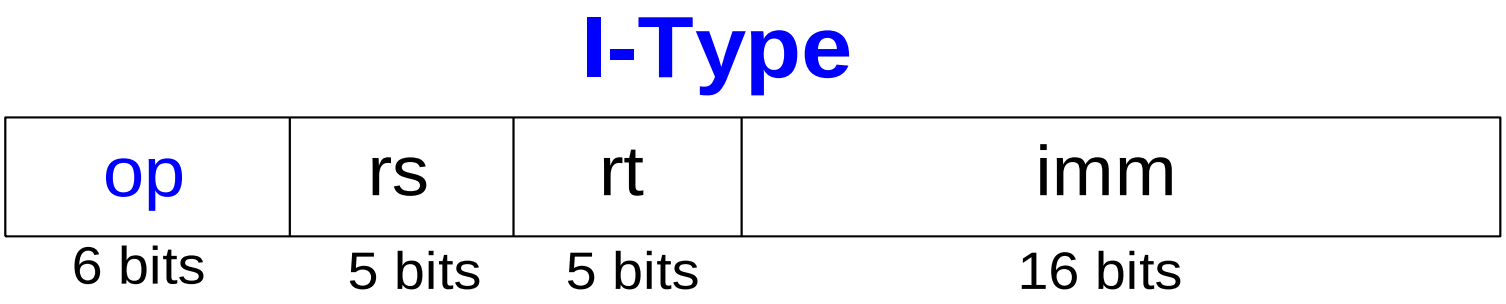
Бележка: редът на регистрите в ***the assembly code*** е:

```
add rd, rs, rt
```

КАРХ: Тема_9: Машинен език. Видове инструкции

I-Type инструкции.

- *Immediate-type*
- Този тип инструкции използва 2 регистъра и една константа като операнди.
- Има 3 операнди:
  - rs, rt: регистри операнди (register operands)
  - imm: 16-bit константа в двоично-допълнителен код (immediate)
- Инструкциите имат 4 полета – op, rs, rt, и imm.
  - op: the opcode
  - Принцип: Simplicity favors regularity: всички инструкции имат *opcode*.
  - Операциите се дефинират напълно и единствено от opcode.
- rs, и imm са винаги източници, rt е получател за някои инструкции или допълнителен източник при други.



КАРХ: Тема_9: Машинен език. Видове инструкции

I-Type инструкции.

Пример:

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

Обърнете внимание на:

редът на регистрите в

assembly и при ***machine codes***:

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

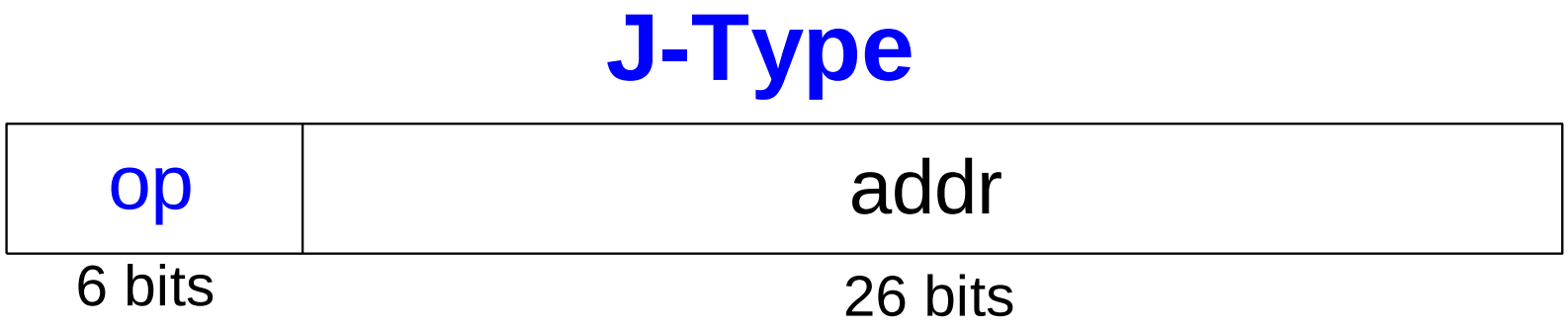
Machine Code

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	

КАРХ: Тема_9: Машинен език. Видове инструкции

J-Type инструкции.

- *Jump-type* инструкции.
- Този формат се използва само в инструкции за преходи и има само един 26-bit адресен операнд (address operand) (`addr`) и полето *opcode*.
- Инструкциите имат 2 полета – *op* и *addr* .



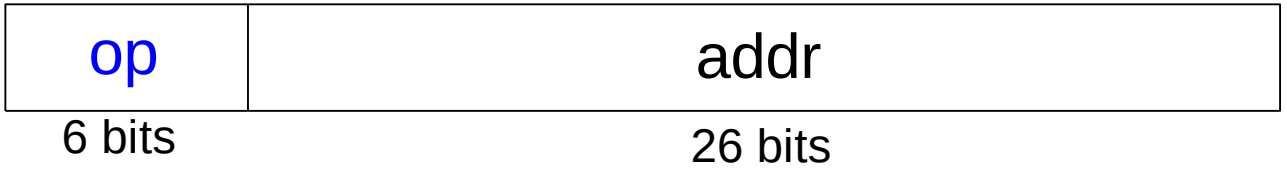
КАРХ: Тема_9: Машинен език. Видове инструкции

Обобщение – формат на инструкциите.

R-Type

I-Type

J-Type



КАРХ: Тема_9: Машинен език. Видове инструкции

Представяне на програмите в компютрите.

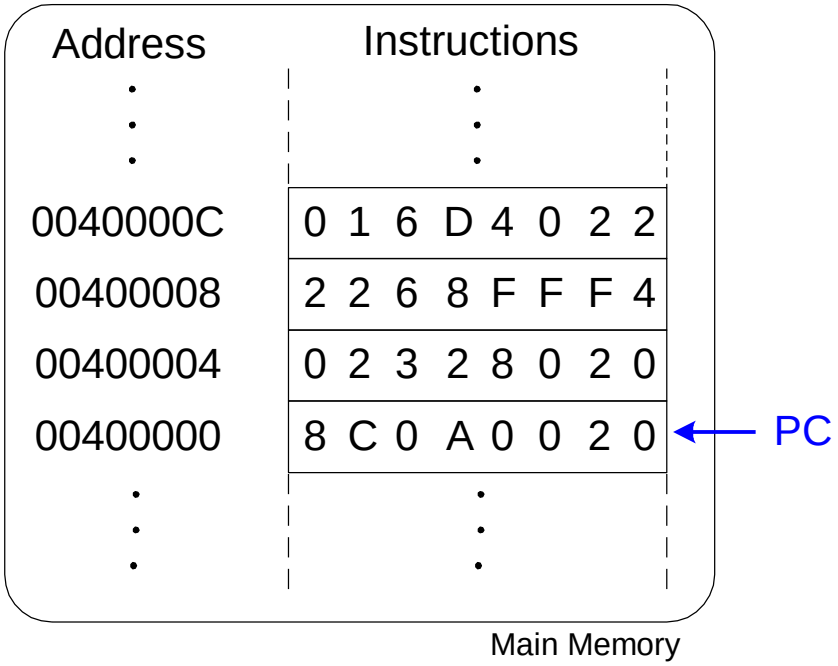
- 32-bit *инструкции* & *данни* се съхраняват в паметта на компютъра.
- Последователността и вида на използваните инструкции е единствената разлика между две приложения (програми).
- За стартиране на нова програма:
  - Не са нужни хардуерни промени в компютъра.
  - Просто се записва новата програма в паметта.
- Изпълнение на програмата:
  - Процесорът взема *(fetches)* (чете) инструкциите последователно от паметта.
  - Процесорът изпълнява указаните в инструкциите операции.

КАРХ: Тема_9: Машинен език. Видове инструкции

Представяне на програмите в компютрите.

Assembly Code	Machine Code
`lw $t2, 32($0)`	`0x8C0A0020`
`add $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub $t0, $t3, $t5`	`0x016D4022`

Stored Program

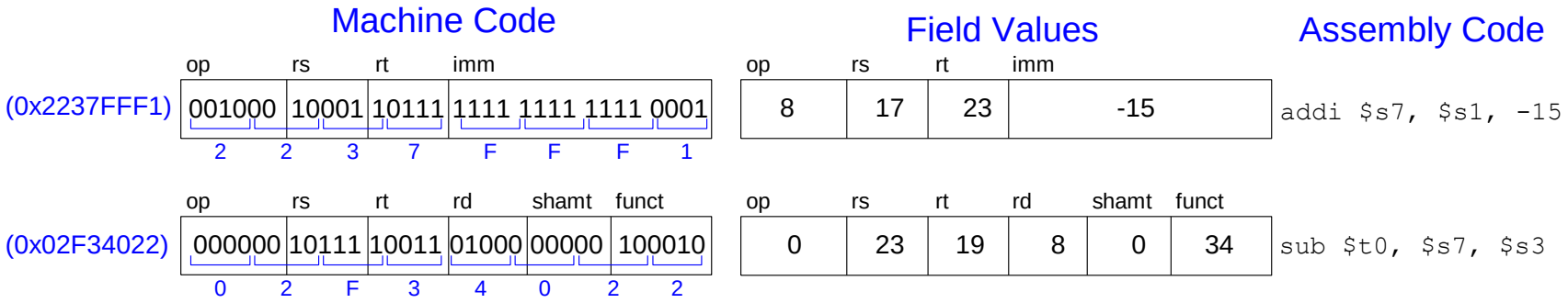


Програмен Брояч (Program Counter) (PC): указва (сочи) към текущата инструкция (изпълнявана в момента).

КАРХ: Тема_9: Машинен език. Видове инструкции

Интерпретация на машинния код.

- Всяка 32-bit инструкция започва с 6-bit opcode , който до голяма степен определя типа на инструкцията.
- Всички инструкции се записват в паметта като 32-bit думи, които оформят програмен код – *програмата*.
- Интерпретацията на машинния код започва с opcode: показва разпределението на останалите полета.
- Ако opcode е 0 :
  - Това R-type инструкция
  - Каква точно е операцията определят Function bits.
- В противен случай:
  - opcode показва каква е операцията.



КАРХ: Тема_9: Машинен език. Видове инструкции

Програмиране.

- Езици от високо ниво:
  - Например: C, Java, Python
  - Написани са на по-високо ниво на абстракция.
- Съдържат програмни конструкции от високо ниво (high-level software constructs):
  - if/else statements (условни преходи)
  - for loops (цикъл с for)
  - while loops (цикъл с while)
  - Arrays (масиви)
  - function calls (извикване на функции, подпрограми)

КАРХ: Тема_9: Машинен език. Видове инструкции

***Ada Lovelace* (1815-1852)**

<u>Програмиране.</u>

- Написала първата компютърна програма.
- Тази програма изчислявала *числата на Бернули* (the Bernoulli numbers) на Аналитичната Машина на Чарлз Бабич (Charles Babbage's Analytical Engine).
- Тя е дъщеря на големия британски поет лорд Байрон (Lord Byron).

КАРХ: Тема_9: Машинен език. Видове инструкции

Използване на логическите инструкции.

- **`and`, `or`, `xor`, `nor`**
  - `and`: полезна за маскиране на битове (**masking** bits)
    - Маскиране на всички байтове без последния:
      0xF234012F AND 0x000000FF = 0x0000002F
  - `or`: полезна за комбиниране (**combining**) на полета от битове:
    - Комбиниране на 0xF2340000 със 0x000012BC:
      0xF2340000 OR 0x000012BC = 0xF23412BC
  - `nor`: полезна за инвертиране (**inverting**) на битове:
    - A NOR $0 = NOT A
- **`andi`, `ori`, `xori`**
  - 16-bit immediate is zero-extended (*not* sign-extended)
  - `nori` не е необходима

КАРХ: Тема_9: Машинен език. Видове инструкции

Използване на логическите инструкции.

Примери:

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code

```
and $s3, $s1, $s2
or  $s4, $s1, $s2
xor $s5, $s1, $s2
nor $s6, $s1, $s2
```

Result

$s3								
$s4								
$s5								
$s6								

КАРХ: Тема_9: Машинен език. Видове инструкции

Използване на логическите инструкции.

Примери:

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code

```
and $s3, $s1, $s2
or  $s4, $s1, $s2
xor $s5, $s1, $s2
nor $s6, $s1, $s2
```

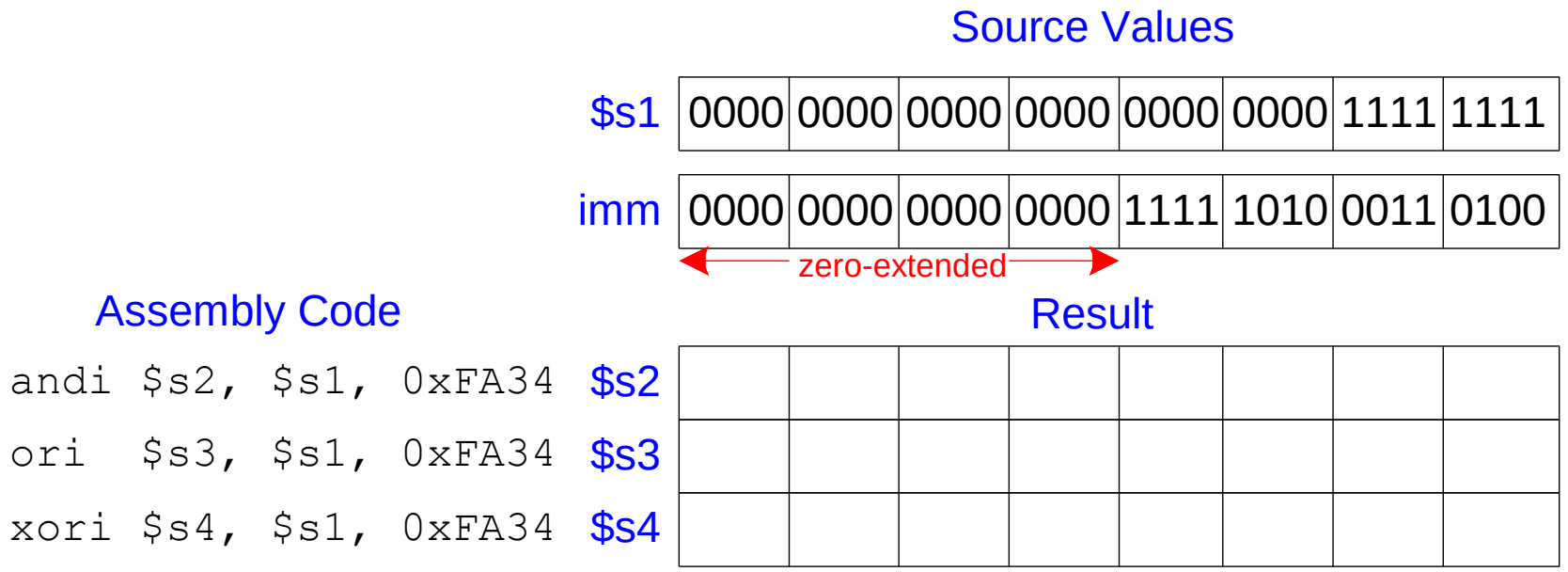
Result

$s3	0100	0110	1010	0001	0000	0000	0000	0000
$s4	1111	1111	1111	1111	1111	0000	1011	0111
$s5	1011	1001	0101	1110	1111	0000	1011	0111
$s6	0000	0000	0000	0000	0000	1111	0100	1000

КАРХ: Тема_9: Машинен език. Видове инструкции

Използване на логическите инструкции.

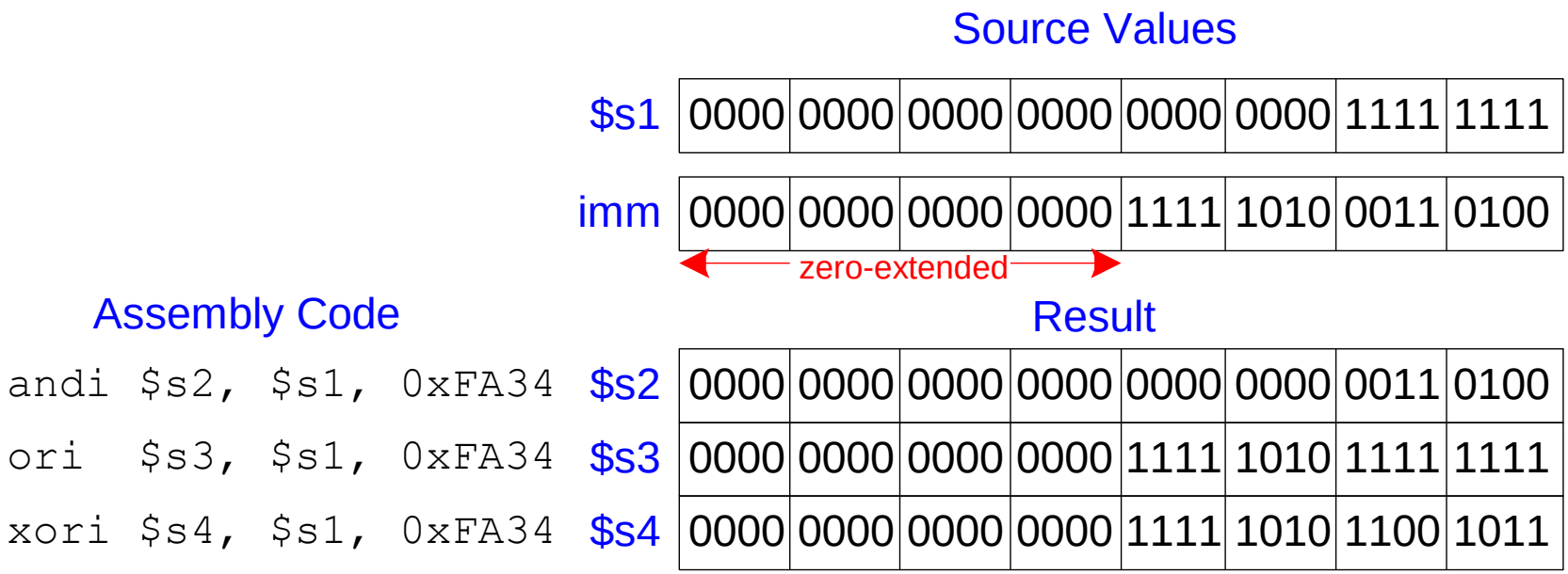
Примери:



КАРХ: Тема_9: Машинен език. Видове инструкции

Използване на логическите инструкции.

Примери:



КАРХ: Тема_9: Машинен език. Видове инструкции

Инструкции за преместване (Shift Instructions).

Преместването е от 1 до 31 bits.

- sll: логическо преместване на ляво (shift left logical)
  - **Пример:** sll $t0, $t1, 5 # $t0 <= $t1 << 5
- srl: логическо преместване на дясно (shift right logical)
  - **Пример:** srl $t0, $t1, 5 # $t0 <= $t1 >> 5
- sra: аритметическо преместване на дясно (shift right arithmetic)
  - **Пример:** sra $t0, $t1, 5 # $t0 <= $t1 >>> 5

КАРХ: Тема_9: Машинен език. Видове инструкции

Инструкции за променливо преместване (Variable Shift Instructions).

Преместването е от 1 до 31 bits. Стойността му се определя от младшите 5 bits на регистъра rs. Формат на инструкциите sllv rd, rt, rs

- sllv: променливо логическо преместване на ляво (shift left logical variable)
  - **Пример:** sllv $t0, $t1, $t2 # $t0 <= $t1 << $t2
- srlv: променливо логическо преместване на дясно (shift right logical variable)
  - **Пример:** srlv $t0, $t1, $t2 # $t0 <= $t1 >> $t2
- srav: променливо аритметическо преместване на дясно (shift right arithmetic variable)
  - **Пример:** srav $t0, $t1, $t2 # $t0 <= $t1 >>> $t2

КАРХ: Тема_9: Машинен език. Видове инструкции

Инструкции за преместване (Shift Instructions).

Пример.

Assembly Code / Field Values

Assembly Code	op	rs	rt	rd	shamt	funct
`sll $t0, $s1, 2`	0	0	17	8	2	0
`srl $s2, $s1, 2`	0	0	17	18	2	2
`sra $s3, $s1, 2`	0	0	17	19	2	3
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	00000	10001	01000	00010	000000	(0x00114080)
000000	00000	10001	10010	00010	000010	(0x00119082)
000000	00000	10001	10011	00010	000011	(0x00119883)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

КАРХ: Тема_9: Машинен език. Видове инструкции

Генериране на константи (Generating Constants).

- 16-bit константи се генерират използвайки `addi`:

C Code

```
// int is a 32-bit signed word
   int a = 0x4f3c;
```

MIPS assembly code

```
# $s0 = a
addi $s0, $0, 0x4f3c
```

- 32-bit константи се генерират чрез load upper immediate (`lui`) и `ori`:

C Code

```
   int a = 0xFEDC8765;
```

MIPS assembly code

```
lui $s0, 0xFEDC
ori $s0, $s0, 0x8765
```

КАРХ: Тема_9: Машинен език. Видове инструкции

Умножение и деление (Multiplication, Division).

- Регистри със специално предназначение (Special registers): lo, hi
- 32 32 умножение, 64 bit резултат
  - mult $s0, $s1
  - Резултата е в {hi, lo}
- 32-bit деление, 32-bit частно, 32-bit остатък
  - div $s0, $s1
  - Частното в lo
  - Остатъка в hi
- Прехвърляне на резултата от специалните регистри lo/hi
  - mflo $s2 (move from low)
  - mfhi $s3 (move from high)