КАРХ: Тема_10: Програмни конструкции от високо ниво

Въведение.

- Разглеждане на 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_10: Програмни конструкции от високо ниво

Преходи (Branching).

- Видове преходи:
  - **Условни (Conditional) – когато условието е изпълнено.**
    - Преход при равенство (branch if equal) (beq)
    - Преход при неравенсрво (branch if not equal) (bne)
  - **Безусловни (Unconditional)**
    - jump (j)
    - jump register (jr)
    - jump and link (jal)

КАРХ: Тема_10: Програмни конструкции от високо ниво

Преход при равенство (beq).

Пример:

```
# MIPS assembly
 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     # условието е изпълнено(branch is taken)
 addi $s1, $s1, 1          # не се изпълнява (not executed)
 sub  $s1, $s1, $s0        # не се изпълнява (not executed)

target:                    # етикет (label)
 add  $s1, $s1, $s0        # $s1 = 4 + 4 = 8
```

Етикетите (Labels) маркират мястото на инструкцията. Те не могат да бъдат запазени думи и завършват с двоеточие (:)

КАРХ: Тема_10: Програмни конструкции от високо ниво

Преход при неравенсрво (bne).

Пример:

```
# MIPS assembly
  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     # условието не е изпълнено
                            #(branch not taken)
  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
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Безусловен преход (jump (j)).

Това е преход към адрес маркиран с етикет.

Пример:

```
# MIPS assembly
  addi $s0, $0, 4              # $s0 = 4
   addi $s1, $0, 1             # $s1 = 1
   j    target                  # преход до етикета target
                                #(jump to target)
   sra  $s1, $s1, 2             # не се изпълнява (not executed)
   addi $s1, $s1, 1             # не се изпълнява (not executed)
   sub  $s1, $s1, $s0           # не се изпълнява (not executed)

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

КАРХ: Тема_10: Програмни конструкции от високо ниво

Безусловен преход (jump register(jr)).

Това е преход към адрес записан в регистър.

Пример:

```
# MIPS assembly
0x00002000        addi $s0, $0, 0x2010
0x00002004        jr   $s0
0x00002008        addi $s1, $0, 1
0x0000200C        sra  $s1, $s1, 2
0x00002010        lw   $s3, 44($s1)
```

`jr` е **R-type** инструкция.

КАРХ: Тема_10: Програмни конструкции от високо ниво

Условно състояние If (If Statement).

- **C Code**

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

     f = f - i;
```

MIPS assembly code

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

Условно състояние If (If Statement).

- **C Code** **MIPS assembly code**

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

```
if (i == j)                bne $s3, $s4, L1
    f = g + h;             add $s0, $s1, $s2

  f = f - i;               L1: sub $s0, $s0, $s3
```

В **Assembly** се проверява обратното условие (`i!=j`) на това, посочено в кода от високо ниво (`i==j`) **!**

КАРХ: Тема_10: Програмни конструкции от високо ниво

Условно състояние If/Else (If/Else Statement).

- **C Code** **MIPS assembly code**

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

КАРХ: Тема_10: Програмни конструкции от високо ниво

Условно състояние If/Else (If/Else Statement).

- **C Code**

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

MIPS assembly code

```
# $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:
```

Цикъл While (While Loops).

- **C Code** **MIPS assembly code**

```
// 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;
}
```

В **Assembly** се проверява обратното условие **(pow == 128)** на това, посочено в C code (**pow != 128**).

Цикъл While (While Loops).

- **C Code** **MIPS assembly code**

```
// determines the power#          $s0 = pow, $s1 = x
// of x such that 2^x = 128
int pow = 1;                      addi $s0, $0, 1
int x   = 0;                      add  $s1, $0, $0
                                  addi $t0, $0, 128
while (pow != 128) {     while:   beq  $s0, $t0, done
  pow = pow * 2;                  sll  $s0, $s0, 1
  x = x + 1;                      addi $s1, $s1, 1
}                                 j    while
                            done:
```

В **Assembly** се проверява обратното условие (**pow == 128**) на това, посочено в C code (**pow != 128**).

КАРХ: Тема_10: Програмни конструкции от високо ниво

Цикъл For (For Loops).

Структура на цикъл **for**

```
for (initialization; condition; loop operation)
    statement
```

- **Initialization(инициализация)**: изпълнява се преди началото на цикъла
- **Condition(условие)**: проверява се в началото на всяка итерация
- **loop operation(брояч)**: изпълнява се в края на всяка итерация
- **Statement(тяло на цикъла)**: изпълнява се всеки път когато условието е изпълнено

КАРХ: Тема_10: Програмни конструкции от високо ниво

Цикъл For (For Loops).

- **C Code**

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
```

Цикъл For (For Loops).

- **C Code**

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
  addi $s1, $0, 0
  add  $s0, $0, $0
  addi $t0, $0, 10
for: beq  $s0, $t0, done
  add  $s1, $s1, $s0
  addi $s0, $s0, 1
  j    for
done:
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Цикъл For с по-малко (For Less Than Loops).

- **C Code** **MIPS assembly code**

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

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

Цикъл For с по-малко (For Less Than Loops).

- **C Code**

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

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

MIPS assembly code

```
      # $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 if i < 101

КАРХ: Тема_10: Програмни конструкции от високо ниво

Масиви(Arrays).

Данни

- Достъп до голямо количество еднотипни данни, организирани като последователни адреси в паметта.
- **Index**: маркира отделен елемент на масива (номер на елемент)
- **Size (размер)** : брой на елементите

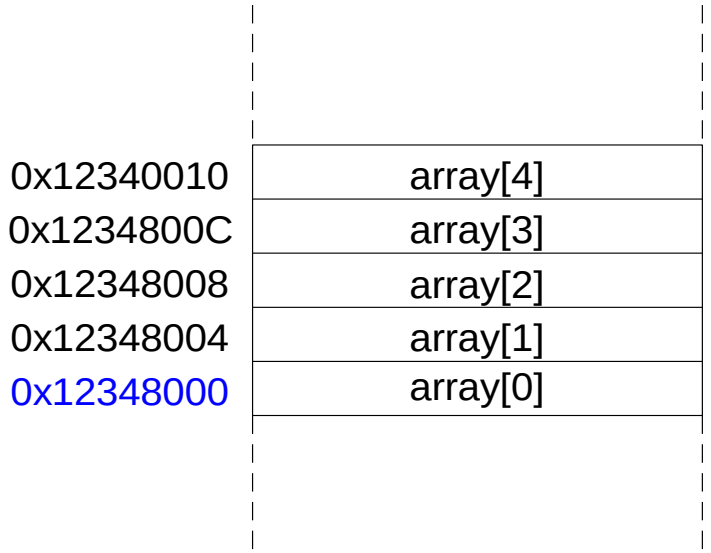
КАРХ: Тема_10: Програмни конструкции от високо ниво

Масиви(Arrays).

- 5-елементен масив
- **Base address** = 0x12348000 (адресът на първия елемент, `array[0]`)
- Първата стъпка за достъп до масива: зареждане на base address в регистър.

// C Code

```
int array[5];
array[0] = array[0] * 2;
array[1] = array[1] * 2;
```



КАРХ: Тема_10: Програмни конструкции от високо ниво

Масиви(Arrays).

```
// C Code
   int array[5];
   array[0] = array[0] * 2;
   array[1] = array[1] * 2;

# MIPS assembly code
# $s0 = array base address
  lui  $s0, 0x1234            # 0x1234 in upper half of $s0
  ori  $s0, $s0, 0x8000       # 0x8000 in lower half of $s0

  lw   $t1, 0($s0)            # $t1 = array[0]
  sll  $t1, $t1, 1            # $t1 = $t1 * 2
  sw   $t1, 0($s0)            # array[0] = $t1

  lw   $t1, 4($s0)            # $t1 = array[1]
  sll  $t1, $t1, 1            # $t1 = $t1 * 2
  sw   $t1, 4($s0)            # array[1] = $t1
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Масиви използвани в цикъл For.

```
// C Code
   int array[1000];
   int i;

   for (i=0; i < 1000; i = i + 1)
        array[i] = array[i] * 8;

# MIPS assembly code
# $s0 = array base address, $s1 = i
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Масиви използвани в цикъл For.

```
# MIPS assembly code
# $s0 = array base address, $s1 = i
# initialization code
  lui  $s0, 0x23B8         # $s0 = 0x23B80000
  ori  $s0, $s0, 0xF000    # $s0 = 0x23B8F000
  addi $s1, $0, 0          # i = 0
  addi $t2, $0, 1000       # $t2 = 1000

loop:
  slt  $t0, $s1, $t2       # i < 1000?
  beq  $t0, $0, done       # if yes then done
  sll  $t0, $s1, 2         # $t0 = i * 4 (byte offset)
  add  $t0, $t0, $s0       # address of array[i]
  lw   $t1, 0($t0)         # $t1 = array[i]
  sll  $t1, $t1, 3         # $t1 = array[i] * 8
  sw   $t1, 0($t0)         # array[i] = array[i] * 8
  addi $s1, $s1, 1         # i = i + 1
  j    loop                # repeat
done:
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

ASCII кодове на символите.

- ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*
- Всеки текстови символ има уникална стойност
- Например, S = 0x53, a = 0x61, A = 0x41
  - Главните и малките букви се различават с 0x20 (32)

КАРХ: Тема_10: Програмни конструкции от високо ниво

ASCII кодове на символите.

Таблица на символите.

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
20	space	30	0	40	@	50	P	60	`	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	'	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o		

КАРХ: Тема_10: Програмни конструкции от високо ниво

Извикване на функции (Function Calls).

- **Caller:** извикващ функцията (в този случай, *main*)
- **Callee:** извикана функция (в този случай, *sum*)

C Code

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Конвенции при функциите.

- **Caller:**
  - изпраща **arguments** на *callee*
  - Прави скок към *callee*
- **Callee:**
  - **Извършва** действията на функцията
  - **Връща** резултат на *caller*
  - **Връща** програмата към точката на повикване
  - **Не трябва да променя стойността** на регистрите или паметта използвана от *caller*

КАРХ: Тема_10: Програмни конструкции от високо ниво

MIPS Конвенции при функциите.

В MIPS архитектурата *caller* използва до 4 аргумента и ги слага в регистрите `$a0÷$a3`, преди да извика *callee,* която използва за върнати стойности регистрите `$v0÷$v1` преди да завърши. Използвайки тази конвенция и двете функции знаят къде ще намерят аргументите си и върнатите стойности. *Caller* запазва адрес за връщане (return address) в регистъра `$ra` и след това извиква *callee* използвайки *jump and link* (`jal`) инструкция. *Callee* **не трябва да променя** памет и регистри използвани от *caller* , т.е. запазващите регистри `$s0÷$s7, $ra` , стекът и частта от паметта, използвана за временни променливи, остават не променени. Връщането от функцията става с инструкцията jump register (`jr`) .

КАРХ: Тема_10: Програмни конструкции от високо ниво

Извикване на функция.

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204 add  $s0, $s1, $s2
..........

0x00401020 simple: jr $ra
```

void означава, че **simple** не връща стойност.

КАРХ: Тема_10: Програмни конструкции от високо ниво

Извикване на функция.

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204 add  $s0, $s1, $s2
..........

0x00401020 simple: jr $ra
```

jal: jumps to `simple`

`$ra` = PC + 4 = 0x00400204

jr $ra: jumps to address in `$ra` (0x00400204)

КАРХ: Тема_10: Програмни конструкции от високо ниво

Аргументи и върната стойност.

C Code

```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}

int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // return value
}
```

Аргументи и върната стойност.

MIPS assembly code

```
# $s0 = y

main:
   ...
   addi $a0, $0, 2     # argument 0 = 2
   addi $a1, $0, 3     # argument 1 = 3
   addi $a2, $0, 4     # argument 2 = 4
   addi $a3, $0, 5     # argument 3 = 5
   jal  diffofsums     # call Function
   add  $s0, $v0, $0   # y = returned value
   ...

# $s0 = result
diffofsums:
   add $t0, $a0, $a1   # $t0 = f + g
   add $t1, $a2, $a3   # $t1 = h + i
   sub $s0, $t0, $t1   # result = (f + g) - (h + i)
   add $v0, $s0, $0    # put return value in $v0
   jr  $ra             # return to caller
```

Аргументи и върната стойност.

MIPS assembly code

```
# $s0 = result
diffofsums:
  add $t0, $a0, $a1   # $t0 = f + g
  add $t1, $a2, $a3   # $t1 = h + i
  sub $s0, $t0, $t1   # result = (f + g) - (h + i)
  add $v0, $s0, $0    # put return value in $v0
  jr  $ra             # return to caller
```

- `diffofsums` променя стойността на 3 регистъра: `$t0`, `$t1`, `$s0`
- `diffofsums` може да използва *stack* за временно запазване на стойностите на тези регистри

КАРХ: Тема_10: Програмни конструкции от високо ниво

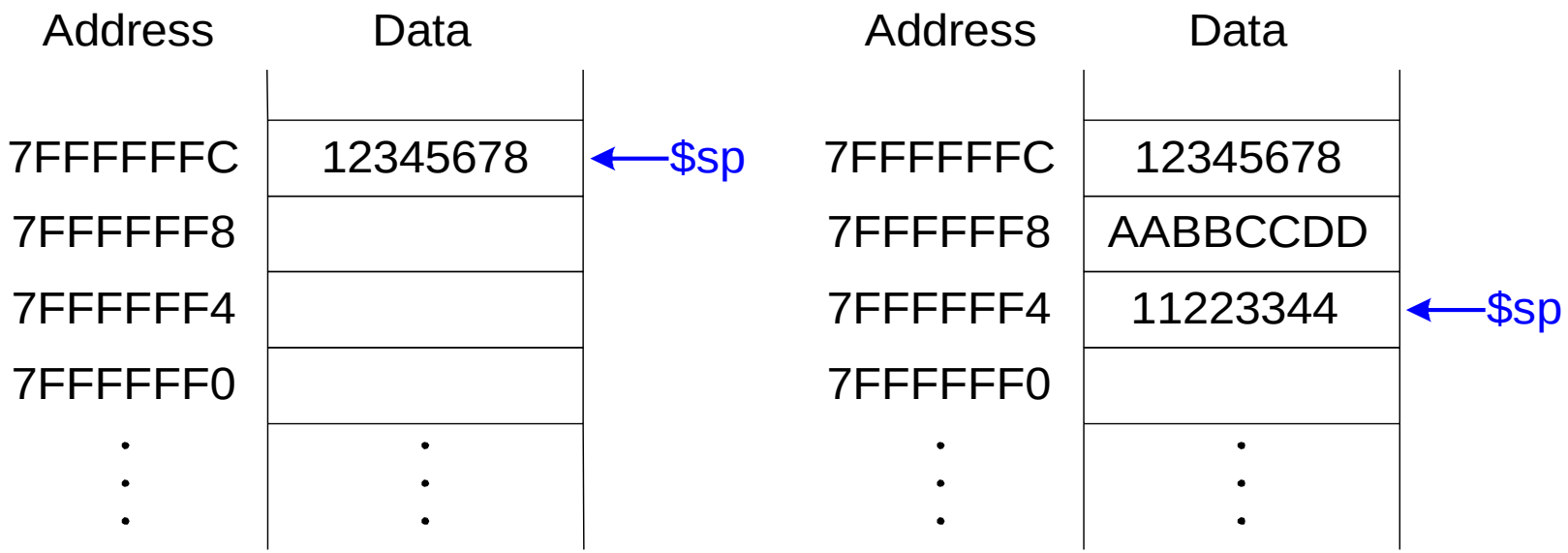
Стекът (The Stack).

- Представлява памет за временно съхраняване на променливи
- Прилича на стълб от чинии и работи на принципа: Последния влязъл – Пръв излязал (last-in-first-out) (LIFO)
- ***Може да се разширява (Expands)*:** използва повече памет, когато има нужда от повече място.
- ***Може да се свива (Contracts)*:** използва по-малко памет, когато вече изполвано място не му е нужно.

КАРХ: Тема_10: Програмни конструкции от високо ниво

Стекът (The Stack).

- Нараства надолу (от по-високи към по-ниски адреси на паметта).
- Stack pointer: $sp сочи към върха на *stack*.



КАРХ: Тема_10: Програмни конструкции от високо ниво

Как функциите използват Стека.

- Извиканите функции не бива да имат нежелани странични ефекти.
- Но `diffofsums` променя стойността на 3 регистъра: `$t0`, `$t1`, `$s0`.

```
# MIPS assembly
# $s0 = result
diffofsums:
  add $t0, $a0, $a1  # $t0 = f + g
  add $t1, $a2, $a3  # $t1 = h + i
  sub $s0, $t0, $t1  # result = (f + g) - (h + i)
  add $v0, $s0, $0   # put return value in $v0
  jr  $ra            # return to caller
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

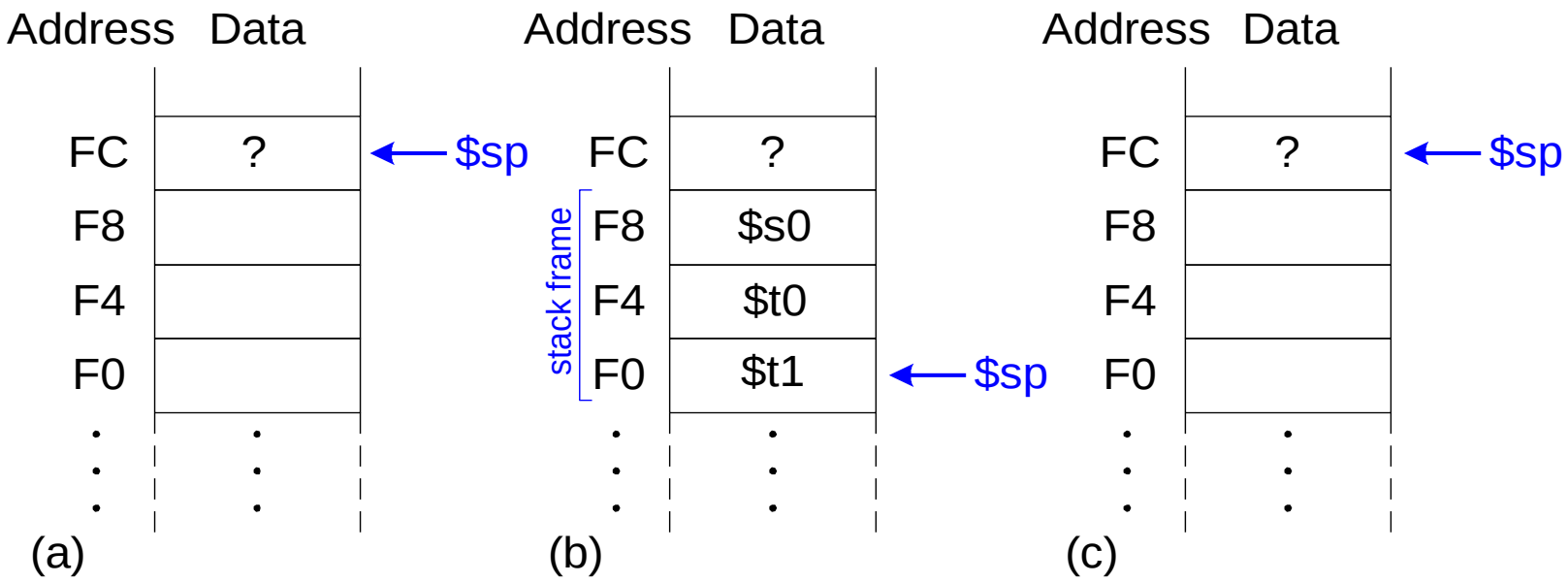
Запазване стойността на регистрите в Стека.

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -12  # make space on stack
                      # to store 3 registers
  sw   $s0, 8($sp)    # save $s0 on stack
  sw   $t0, 4($sp)    # save $t0 on stack
  sw   $t1, 0($sp)    # save $t1 on stack
  add  $t0, $a0, $a1  # $t0 = f + g
  add  $t1, $a2, $a3  # $t1 = h + i
  sub  $s0, $t0, $t1  # result = (f + g) - (h + i)
  add  $v0, $s0, $0   # put return value in $v0
  lw   $t1, 0($sp)    # restore $t1 from stack
  lw   $t0, 4($sp)    # restore $t0 from stack
  lw   $s0, 8($sp)    # restore $s0 from stack
  addi $sp, $sp, 12   # deallocate stack space
  jr   $ra            # return to caller
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Запазване стойността на регистрите в Стека.

Пример: `diffofsums`



КАРХ: Тема_10: Програмни конструкции от високо ниво

Запазени и незапазени регистри.

Preserved *Callee-Saved*	Nonpreserved *Caller-Saved*
`$s0-$s7`	`$t0-$t9`
`$ra`	`$a0-$a3`
`$sp`	`$v0-$v1`
stack above `$sp`	**stack below `$sp`**

КАРХ: Тема_10: Програмни конструкции от високо ниво

Обобщение.

- **Caller**
  - Задава стойности на аргументите в $a0-$a3
  - Запазва всички необходими регистри ($ra, вероятно и $t0-t9)
  - jal callee
  - Възстановява регистрите
  - Търси резултата в $v0
- **Callee**
  - Запазва регистрите, които може да засегне ($s0-$s7)
  - Изпълнява зададената функция
  - Поставя резултата в $v0
  - Възстановява стойностите на регистрите
  - jr $ra

КАРХ: Тема_10: Програмни конструкции от високо ниво

Методи на адресация (Addressing Modes).

Как се адресират операндите?

- Регистърно (Register Only)
- Непостедствено (Immediate)
- Чрез базов адрес (Base Addressing)
- Относително програмния брояч (PC-Relative)
- Псевдо-директно (Pseudo Direct)

КАРХ: Тема_10: Програмни конструкции от високо ниво

Методи на адресация (Addressing Modes).

Регистърно (Register Only)

- Операндите се намират в регистрите
  - **Пример:** add $s0, $t2, $t3
  - **Пример:** sub $t8, $s1, $0

Непосредствено (Immediate)

- 16-bit immediate са използват като операнди
  - **Пример:** addi $s4, $t5, -73
  - **Пример:** ori $t3, $t7, 0xFF

Чрез базов адрес (Base Addressing)

- Адресът на операнда е:

  Базов адрес + знаково-продължена константа (sign-extended immediate)
  - **Пример:** lw $s4, 72($0)
    - адрес = $0 + 72
  - **Пример:** sw $t2, -25($t1)
    - адрес = $t1 - 25

Методи на адресация (Addressing Modes).

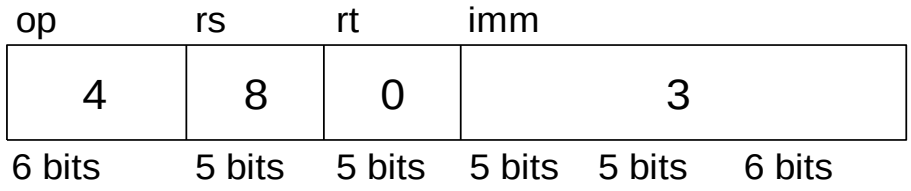
- **Относително програмния брояч (PC-Relative Addressing)**

```
0x10          beq   $t0, $0, else
0x14          addi  $v0, $0, 1
0x18          addi  $sp, $sp, i
0x1C          jr    $ra
0x20    else: addi  $a0, $a0, -1
0x24          jal   factorial
```

Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

Field Values



КАРХ: Тема_10: Програмни конструкции от високо ниво

Методи на адресация (Addressing Modes).

- **Псевдо-директно адресиране (Pseudo-direct Addressing)**

```
0x0040005C          jal     sum
...
0x004000A0  sum:    add     $v0, $a0, $a1
```

JTA 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

26-bit addr 0000 0000 0100 0000 0000 0000 1010 0000 (0x0100028)

0 1 0 0 0 2 8

Field Values

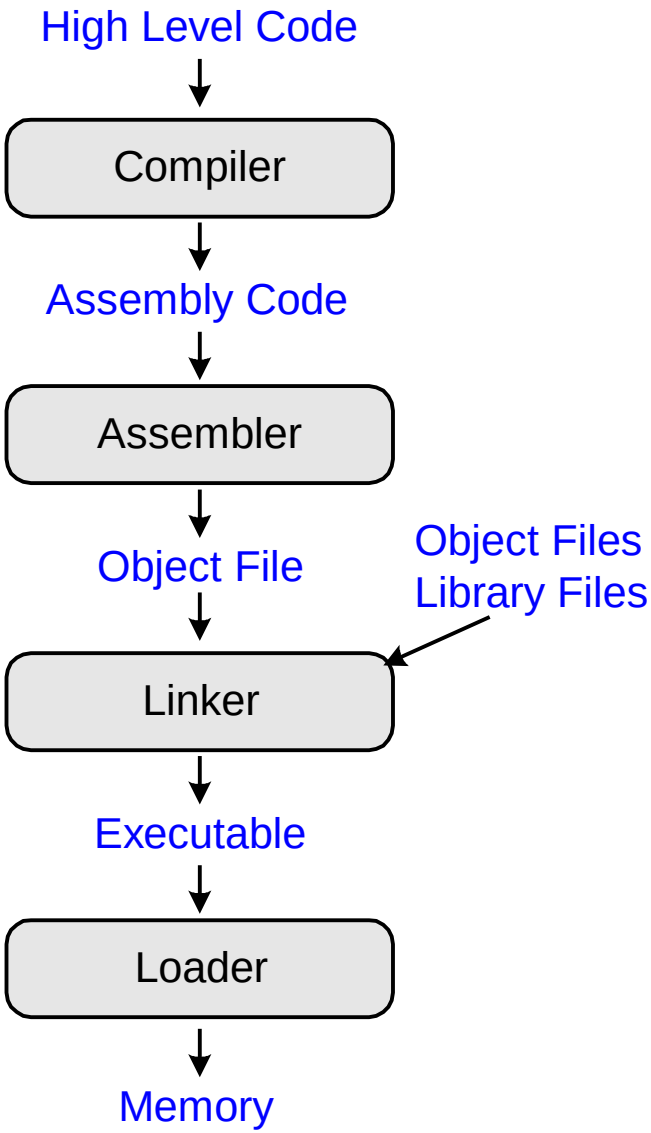
op	imm
3	0x0100028
6 bits	26 bits

Machine Code

op	addr	
000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
6 bits	26 bits	

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.



КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Grace Hopper, 1906-1992

- Дипломирала се в Yale University с докторска степен (Ph.D.) по математика
- Разработила първия компилатор
- Участвала в разработката на програмния език COBOL
- Високо награждаван морски офицер
- Получила **World War II Victory Medal** и **National Defense Service Medal** и много други военни отличия

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

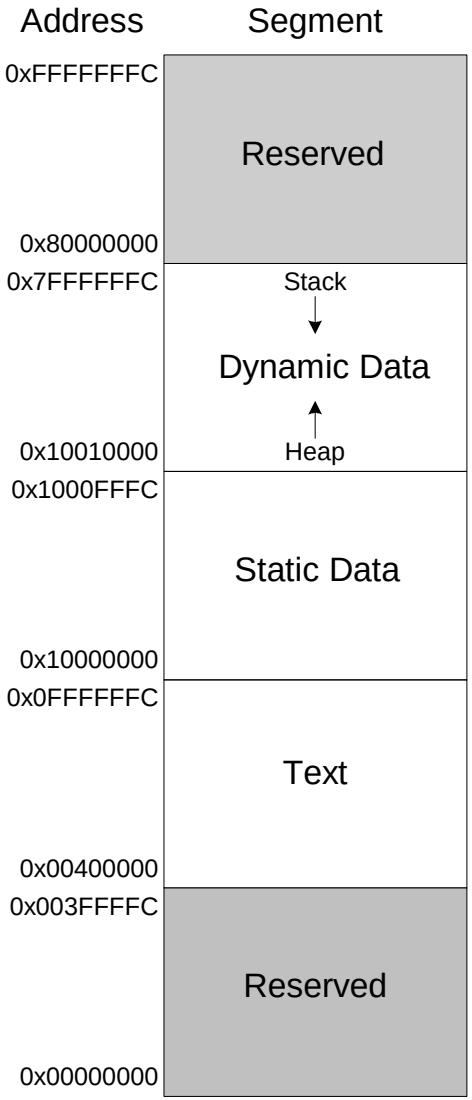
Какво се съхранява в паметта?

- Инструкции (Instructions) (наричани също *text*)
- Данни (Data)
  - Общи/статични (Global/static): налични в паметта преди започването на програмата
  - Динамични (Dynamic): получени при изпълнението на програмата
- Колко е голяма паметта?
  - Точно 2^{32} = 4 gigabytes (4 GB)
  - От адрес **0x00000000** до **0xFFFFFFFF**

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Разпределение (карта) на паметта в MIPS (MIPS Memory Map).



КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Примерна програма. C Code

```
int f, g, y;  // global variables

int main(void)
{
  f = 2;
  g = 3;
  y = sum(f, g);

  return y;
}



int sum(int a, int b) {
  return (a + b);
}
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Примерна програма. MIPS Assembly

```
.data
f:
g:
y:
.text
main:
  addi $sp, $sp, -4   # stack frame
  sw   $ra, 0($sp)    # store $ra
  addi $a0, $0, 2     # $a0 = 2
  sw   $a0, f         # f = 2
  addi $a1, $0, 3     # $a1 = 3
  sw   $a1, g         # g = 3
  jal  sum            # call sum
  sw   $v0, y         # y = sum()
  lw   $ra, 0($sp)    # restore $ra
  addi $sp, $sp, 4    # restore $sp
  jr   $ra            # return to OS
sum:
  add  $v0, $a0, $a1  # $v0 = a + b
  jr   $ra            # return
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Символна таблица.

Symbol	Address
`f`	
`g`	
`y`	
`main`	
`sum`	

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Символна таблица.

Symbol	Address
f	**0x10000000**
g	**0x10000004**
y	**0x10000008**
main	**0x00400000**
sum	**0x0040002C**

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Изпълним код.

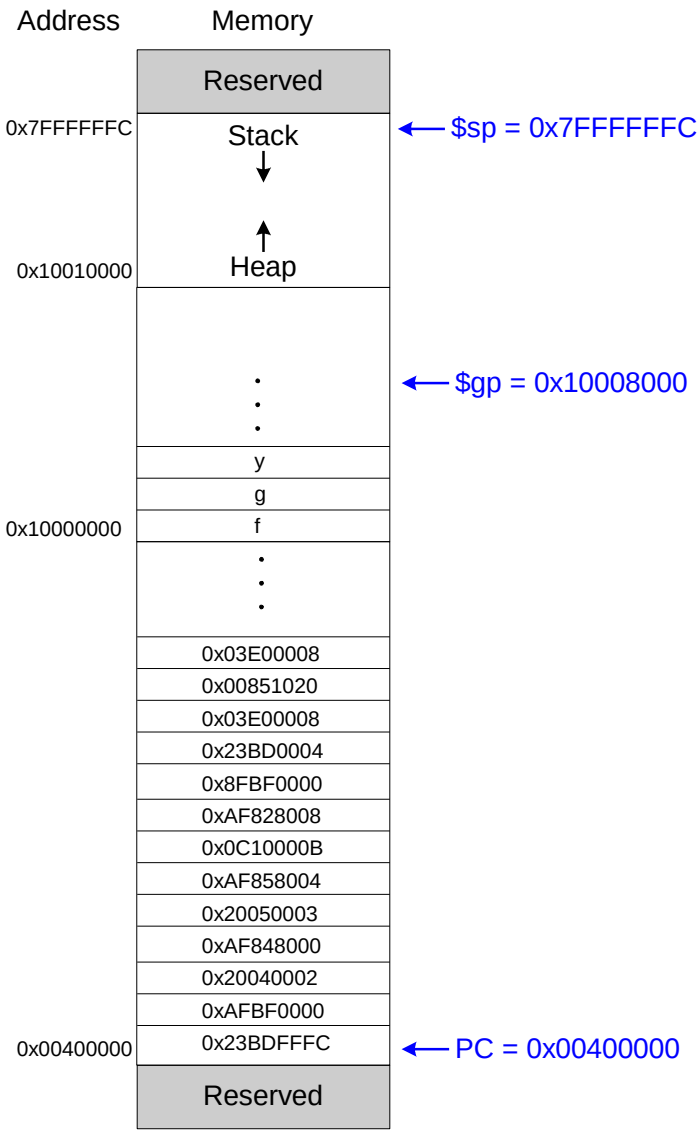
Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	**Address**	**Instruction**
	0x00400000	0x23BDFFFC
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E00008
Data segment	**Address**	**Data**
	0x10000000	f
	0x10000004	g
	0x10000008	y

```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```

КАРХ: Тема_10: Програмни конструкции от високо ниво

Компилиране и стартиране на програмите.

Разположение в паметта.



КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Псевдо-инструкции (Pseudoinstructions)
- Изключения (Прекъсвания) (Exceptions)
- Инструкции със и без знак (Signed and unsigned instructions)
- Инструкции с плаваща запетая (Floating-point instructions)

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Псевдо-инструкции (Pseudoinstructions)

Pseudoinstruction	MIPS Instructions
`li $s0, 0x1234AA77`	`lui $s0, 0x1234` `ori $s0, 0xAA77`
`clear $t0`	`add $t0, $0, $0`
`move $s1, $s2`	`add $s2, $s1, $0`
`nop`	`sll $0, $0, 0`

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Изключения (Прекъсвания) (Exceptions)

Нередовна (невалидна) операция предизвиква преход към програма за обработка на прекъсванията (*exception handler*)

Прекъсванията се причиняват от:

- Hardware *interrupt*, напр. от клавиатурата
- Software *traps*, напр. от недефинирана инструкция

Когато възникне прекъсване, процесорът:

- Записва причината за прекъсването
- Прави преход към *exception handler* (at instruction address 0x80000180)
- След обработката на прекъсването се връща към прекъснатата програма.

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Изключения (Прекъсвания) (Exceptions)

Регистри на прекъсванита (Exception Registers).

Не са част от регистър файла

- **`Cause`**: Записва причината за прекъсването
- **`EPC`** (Exception PC): Записва стойността на PC в мястото на прекъсването

`EPC` и `Cause` са част от *Coprocessor 0*

Запис на данни от *Coprocessor 0* (Move from Coprocessor 0)

- `mfc0 $k0, EPC`

Прехвърля съдържанието на `EPC` в `$k0`

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Изключения (Прекъсвания) (Exceptions)

Причини за прекъсванията.

Exception	Cause
Hardware Interrupt	0x00000000
System Call	0x00000020
Breakpoint / Divide by 0	0x00000024
Undefined Instruction	0x00000028
Arithmetic Overflow	0x00000030

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Изключения (Прекъсвания) (Exceptions)

Обработка на прекъсването.

- Процесорът записва причината и exception PC в `Cause` и `EPC`
- Процесорът прави преход към *exception handler* (0x80000180)
- Exception handler:
  - Записва регистрите в стека
  - Прочита `Cause` register

    `mfc0 $k0, Cause`
  - Обработва прекъсването
  - Възстановява регистрите
  - Прави връщане към прекъснатата програма
    - `mfc0 $k0, EPC`
    - `jr $k0`

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции със и без знак (Signed and unsigned instructions)
  - Събиране и изваждане
  - Умножение и деление
  - Set less than

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции със и без знак (Signed and unsigned instructions)

  - Събиране и изваждане

  Signed: `add`, `addi`, `sub`

  – Същите операции като unsigned versions

  – Процесорът прави прекъсване при *overflow*

  Unsigned: `addu`, `addiu`, `subu`

  – Не се прави прекъсване при *overflow*

  – **Бележка:** `addiu` sign-extends the immediate

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции със и без знак (Signed and unsigned instructions)

  - Умножение и деление

  Signed: `mult`, `div`

  Unsigned: `multu`, `divu`

  - Set less than

  Signed: `slt`, `slti`

  Unsigned: `sltu`, `sltiu`

Бележка: `sltiu` sign-extends the immediate before comparing it to the register

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции за зареждане (четене) със и без знак (Signed and unsigned Loads)

Signed:

- Sign-extends to create 32-bit value to load into register
- Load halfword: `lh`
- Load byte: `lb`

Unsigned:

- Zero-extends to create 32-bit value
- Load halfword unsigned: `lhu`
- Load byte: `lbu`

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции с плаваща запетая (Floating-point instructions)

  Floating-point coprocessor (Coprocessor 1)

  Тридесет и два 32-bit floating-point registers (`$f0-$f31`)

  Числата с двойна точност се записват в два floating point registers
  - Напр., `$f0` и `$f1`, `$f2` и `$f3`, и т.н.
  - Double-precision floating point registers: `$f0`, `$f2`, `$f4`, и т.н.

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции с плаваща запетая (Floating-point instructions)

  Floating-point registers

Name	Register Number	Usage
`$fv0 - $fv1`	0, 2	return values
`$ft0 - $ft3`	4, 6, 8, 10	temporary variables
`$fa0 - $fa1`	12, 14	Function arguments
`$ft4 - $ft8`	16, 18	temporary variables
`$fs0 - $fs5`	20, 22, 24, 26, 28, 30	saved variables

КАРХ: Тема_10: Програмни конструкции от високо ниво

Остатъци (допълнения) (Odds & Ends).

- Инструкции с плаваща запетая (Floating-point instructions)

  Floating-point (F-Type) Instruction Format

- `Opcode` = 17 (010001_2)
- Single-precision:
  - `cop` = 16 (010000_2)
  - `add.s`, `sub.s`, `div.s`, `neg.s`, `abs.s`, etc.
- Double-precision:
  - `cop` = 17 (010001_2)
  - `add.d`, `sub.d`, `div.d`, `neg.d`, `abs.d`, etc.
- 3 register operands:
  - `fs`, `ft`: source operands
  - `fd`: destination operands

F-Type