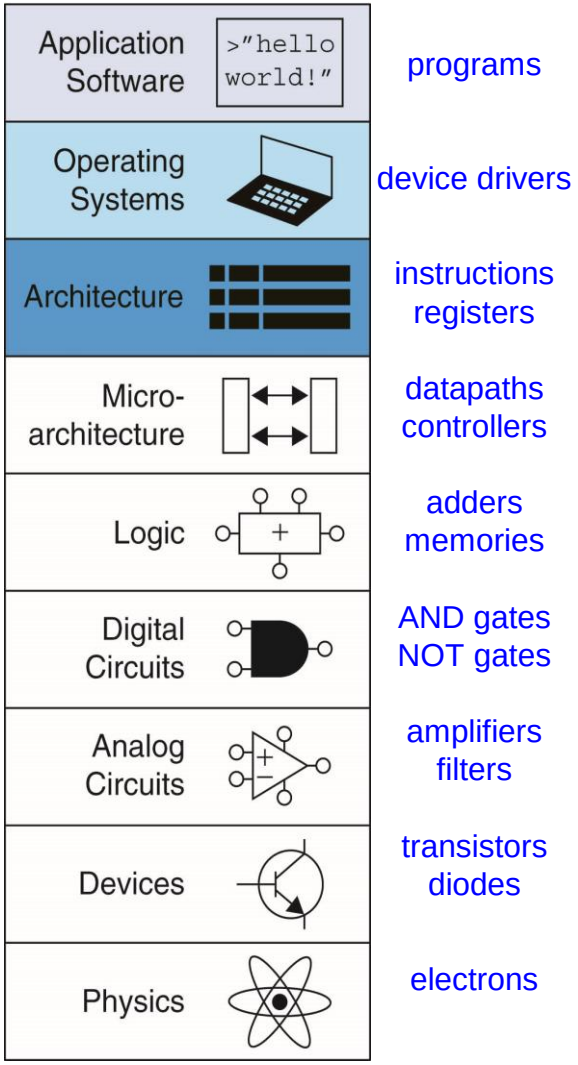
КАРХ: Тема_13: ARM архитектура

Въведение.

- **Архитектура на компютъра –** това е погледа (визията) на програмиста за компютъра.
- Дефинира се чрез набора от инструкции (езика) и мястото на операндите (регистри и памет).
- Съществуват множество различни архитектури – x86, MIPS, SPARC, Power PC.
- Първа стъпка към разбирането на архитектурата на компютъра е научаването на **неговия език**.
- Думите на комп. език – **инструкции**, речникът му – набор от инструкции (**instruction set**).
- Инструкцията представлява операция, която се извършва върху дадени обекти – **операнди.**
- Операнди могат да бъдат регистри на процесора, адреси на клетки от паметта, константи (числа).



КАРХ: Тема_13: ARM архитектура

Въведение.

- Инструкциите се кодират с двоични числа – **машинен език**.
- Представянето на инструкциите в символен формат – **асемблерен език** (асемблер) на процесора.
- Наборът от инструкции на различните архитектури е по-скоро различен диалект, отколкото различен език.
- Компютърната архитектура не определя какъв точно да е хардуерът за дадена реализация, т.е. възможни са много хардуерни реализации, описващи дадена архитектура.
- **Микроархитектура:** специфичният набор от регистри, памети, АЛУ и други изграждащи блокове при направата на даден микропроцесор.
- В дадена архитектура могат да съществуват множество различни микроархитектури.
- Запознаване с **ARM** архитектурата:
  - Разработена от Acorn Computer Group (чрез дъщерната компания Advanced RISC Machines Ltd., по-известна като ARM през 80-те години на миналия век).
  - Над 10 молиарда ARM процесора се продават всяка година (основно в клетъчните телефони и таблетите). Намират приложение в игралните автомати, мобилните камери, в роботиката, автомобилостроенето и др.
  - Компанията ARM не произвежда директно процесори, а продава лиценз на много фирми да произвеждат и продават ARM процесори в продуктите си, като Samsung, Altera, Apple, Cualcomm и др.
  - Представяне на основните инструкции, локациите на операндите и форматът им на машинен език.

КАРХ: Тема_13: ARM архитектура

Общи принципи при ARM архитектурата.

- Всяка инструкция специфицира както операцията, която се извършва, така и операндите, върху които се извършва.
- Наборът от инструкции в ARM архитектурата съдържа само прости, често използващи се инструкции, като броят им се поддържа малък.
- Така хардуерът, който ги изпълнява, да е прост, малък и бърз.
- Всички по-сложни операции се представят чрез поредица от прости инструкции. За това ARM архитектурата спада към групата на т. нар. RISC (Reduced Instruction Set Computers) архитектури. (Има и CISC архитектури).
- Малкият набор инструкции позволява лесното им кодиране и декодиране, например за 64 инструкции са необходими $\log_2 64 = 6$ bit за кодиране.
- ARM е 32 bit архитектура, защото оперира с 32 bit данни (има и 64 bit версия ARMv8).
- Операндите са регистри, памет и константи.
- ARM архитектурата използва 16 регистъра (набор регистри, регистър файл).
- Регистър файлът най-често е изграден от малка SRAM памет с декодер, адресиращ всяка клетка от паметта.

КАРХ: Тема_13: ARM архитектура

Регистри в ARM архитектурата.

- ARM регистрите се означават с R, напр. R1 означава регистър R1.
- При ARM данни могат да се запазват в 13 от 16-те регистъра, а именно R0÷R12.
- Предполага се познаване на някои от езиците от високо ниво като C, C++ или Java.
- Примери – събиране и изваждане:

C Code	ARM assembly code
`a = b + c;`	`add a, b, c`

- **add:** мнемоничен код на операцията;
- **b, c: операнди източници** (source operands);
- **a: операнд получател** (destination operand) (където се записва резултата).
- В езика C инструкциите завършват с (;), // е коментар в една линия, а /* е коментар в много линии */
- При ARM коментарът е само в една линия и се бележи с (;).

C Code	ARM assembly code
`a = b - c;`	`sub a, b, c`

КАРХ: Тема_13: ARM архитектура

Регистри в ARM архитектурата.

- По-сложните операции изискват няколко ARM инструкции.

C Code	ARM assembly code
`a = b + c - d;`	`add t, b, c  ; t = b + c` `sub a, t, d  ; a = t - d`

- В действителност ARM кодът трябва да изглежда така:

C Code	ARM assembly code
`a = b + c;`	`; R0 = a, R1 = b, R2 = c` `add R0, R1, R2  ; a = b + c`

`a = b + c - d;`	`; R3 = d, R4 = t`
	`add R4, R1, R2  ; t = b + c`
	`sub R0, R4, R3  ; a = t - d`

КАРХ: Тема_13: ARM архитектура

Набор регистри в ARM архитектурата.

Name	Use
R0	Argument / return value / temporary variable
R1÷R3	Argument / temporary variables
R4÷R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

КАРХ: Тема_13: ARM архитектура

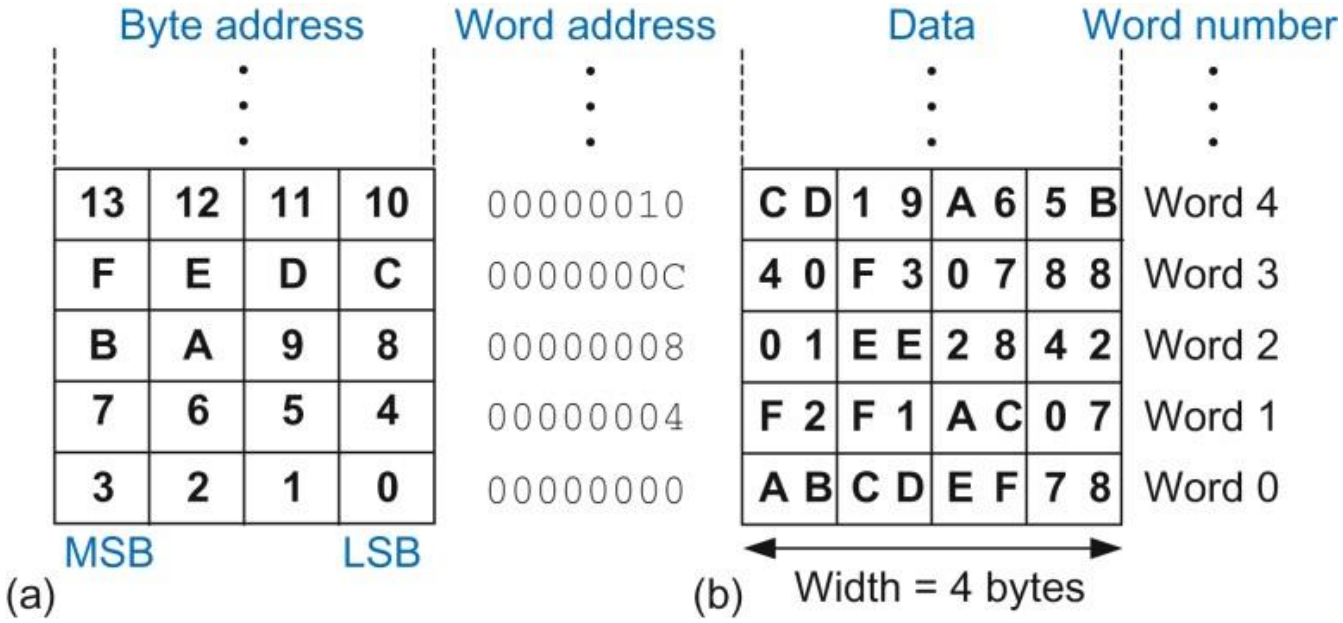
Памет в ARM архитектурата.

- Данните в компютъра са твърде много за да се поберат само в 16 регистъра.
- Повечето данни се пазят в паметта (голяма, но бавна).
- В ARM архитектурата инструкциите използват изключително регистрите, за това данните се прехвърлят от паметта в регистрите.
- ARM архитектурата използва 32 bit адреси за 32 bit думи, като адресирането е на байтове, т.е. всеки байт има уникален адрес.
- За начало разглеждаме адресирането на цели думи (4 байта).

a) адрес на байтовете

b) адрес на данни

Например, адрес 4 съдържа данните 0xF2F1AC07



КАРХ: Тема_13: ARM архитектура

Четене на данни от паметта.

- Четенето на данни от паметта се нарича *зареждане* ***(load).***
- **Мнемоничен код :** *load register* (LDR)
- **Формат:**

  LDR R7, [R5, #8]
- **Изчисляване на адреса:**
  - Събира се базовия адрес *(base register)* (R5) с отместването *(offset)*(8)
  - Реален адрес (address) = (R5 + 8)
- **Резултат:**
  - Регистърът R7 съдържа стойността (данните) записани на адрес (R5 + 8)

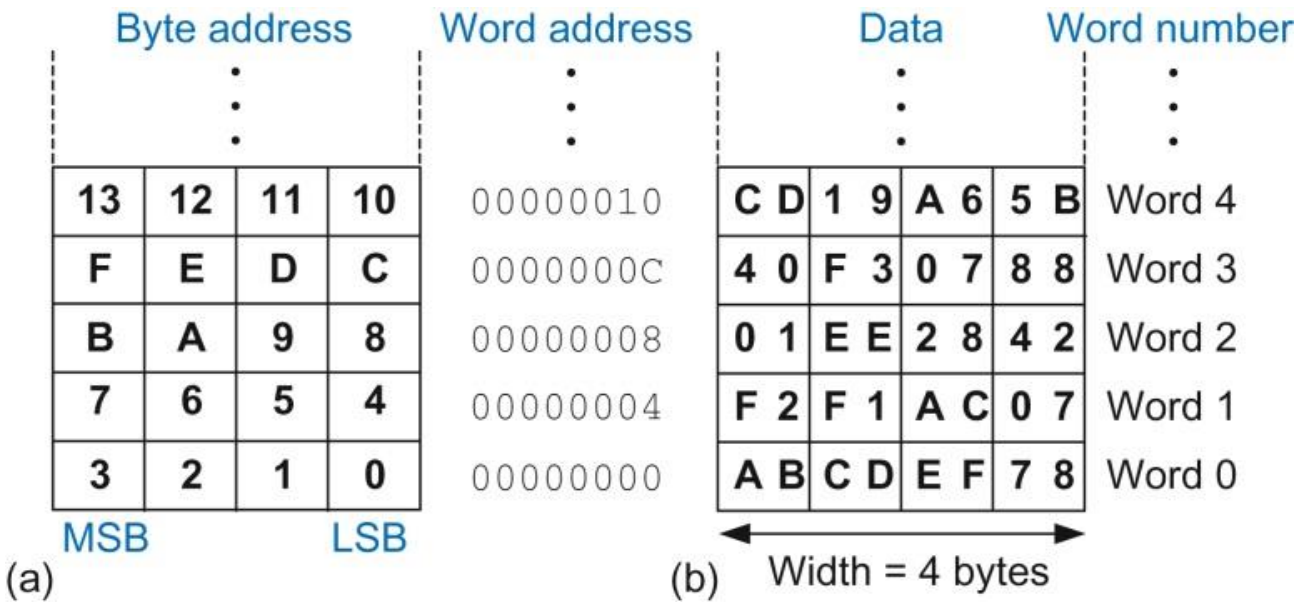
КАРХ: Тема_13: ARM архитектура

Четене на данни от паметта.

- **Пример:** зареждане на регистъра R7 със съдържанието на думата записана на адрес 8.
  - address = (0 + 8) = 8
  - R7= 0x01EE2842 след зареждането

- **High-Level Code** **ARM Assembly code**

```
a = mem[2];          ; R7 = a
                       MOV R5, #0   ; base address = 0
                       LDR R7, [R5, #8] ; R7 <= data at(R5+8)
```



КАРХ: Тема_13: ARM архитектура

Запис на данни в паметта.

- Записът в паметта се нарича съхраняване ***(store).***
- **Мнемоничен код :** *store register* (STR)
- **Пример:** Запис (store) на стойността (42) на регистъра R9 в паметта на адрес 5.
  - Събира се базовия адрес *(base address)*(R1) с отместването *(offset)*(0x14)
  - Реален адрес (address) : (0 + 0x14) = 20
  - Отместването (Offset) може да е десетично число /по подразбиране (default)/ или шестнадесетично число (hexadecimal)

High-Level Code

```
mem[5] = 42
```

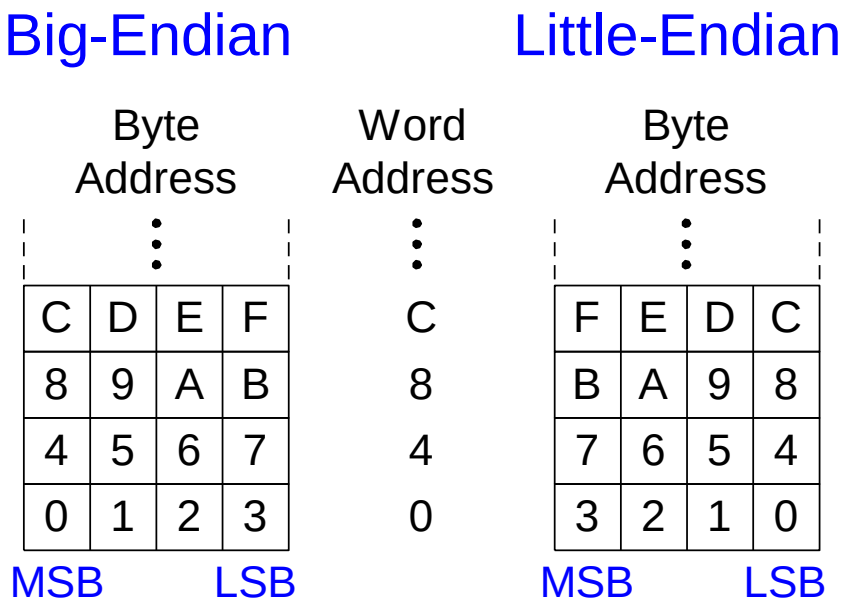
ARM Assembly code

```
MOV R1, #0          ; base address = 0
MOV R9, #42
STR R9, [R1, #0x14] ; value stored at
; memory address (R1+20) = 42
```

КАРХ: Тема_13: ARM архитектура

Организация на адресиране на данните в паметта.

- Как се номерират байтовете в една дума?
- **Little-endian:** номерирането на байтовете започва от малкия (least significant) край.
- **Big-endian:** номерирането на байтовете започва от големия (most significant) край.
- **Адресът на думата** е **един и същ** за big- или little-endian.
- ARM архитектурата предпочита little-endian, но поддържа в някои версии и другия формат (bi-endian).



КАРХ: Тема_13: ARM архитектура

Константи/непосредствени операнди *(immediates).*

- Стойностите им се задават непосредствено, без да е необходим достъп до регистър или памет.
- Използване – напр. в инструкцията **MOV** ;

High-Level Code

```
i = 0;
x = 4080;
```

ARM assembly code

```
; R4 = i, R5 = x
  MOV R4, #0       ; i = 0
  MOV R5, #0xFF0   ; x = 4080
```

- Константите се предхождат от символа (#) и могат да бъдат в десетичен или шестнадесетичен формат.
- Константата е от 8- до 12-bit число без знак.

КАРХ: Тема_13: ARM архитектура

Използване на логическите инструкции.

- **AND, ORR(OR), EOR(XOR), BIC** – оперират побитово върху операндите – източници и записват резултата в регистъра-цел. Първият източник винаги е регистър, а вторият – константа или регистър.
  - BIC (bit clear): полезна за маскиране на битове (**masking** bits)
    - Напр. BIC R6, R1, R2 изчислява R1 AND NOT R2 и записва резултата в R6
  - ORR: полезна за комбиниране (**combining**) на полета от битове:
    - Комбиниране на 0xF2340000 със 0x000012BC:

      0xF2340000 ORR 0x000012BC = 0xF23412BC
- **MVN(MoVe and NOT)** – извършва побитово not (**inverting**) върху втория източник (константа или регистър и записва резултата в регистъра-цел.

КАРХ: Тема_13: ARM архитектура

Използване на логическите инструкции.

Примери:

	Source registers			
R1	0100 0110	1010 0001	1111 0001	1011 0111
R2	1111 1111	1111 1111	0000 0000	0000 0000

Assembly code		Result			
AND R3, R1, R2	R3	0100 0110	1010 0001	0000 0000	0000 0000
ORR R4, R1, R2	R4	1111 1111	1111 1111	1111 0001	1011 0111
EOR R5, R1, R2	R5	1011 1001	0101 1110	1111 0001	1011 0111
BIC R6, R1, R2	R6	0000 0000	0000 0000	1111 0001	1011 0111
MVN R7, R2	R7	0000 0000	0000 0000	1111 1111	1111 1111

КАРХ: Тема_13: ARM архитектура

Инструкции за преместване (Shift Instructions).

Преместването е от 1 до 31 bits.

- LSL: логическо преместване на ляво (logical shift left)
- LSR: логическо преместване на дясно (logical shift right)
- ASR: аритметическо преместване на дясно (arithmetic shift right)
- ROR: ротация на дясно (rotate right)
- ROL инструкция не е необходима.

	Source register			
R5	1111 1111	0001 1100	0001 0000	1110 0111

Assembly Code		Result			
LSL R0, R5, #7	R0	1000 1110	0000 1000	0111 0011	1000 0000
LSR R1, R5, #17	R1	0000 0000	0000 0000	0111 1111	1000 1110
ASR R2, R5, #3	R2	1111 1111	1110 0011	1000 0010	0001 1100
ROR R3, R5, #21	R3	1110 0000	1000 0111	0011 1111	1111 1000

КАРХ: Тема_13: ARM архитектура

Инструкции за преместване (Shift Instructions).

Стойността на преместването може да е записана в регистър.

	Source registers			
R8	0000 1000	0001 1100	0001 0110	1110 0111
R6	0000 0000	0000 0000	0000 0000	0001 0100

Assembly code		Result			
`LSL R4, R8, R6`	R4	0110 1110	0111 0000	0000 0000	0000 0000
`ROR R5, R8, R6`	R5	1100 0001	0110 1110	0111 0000	1000 0001

КАРХ: Тема_13: ARM архитектура

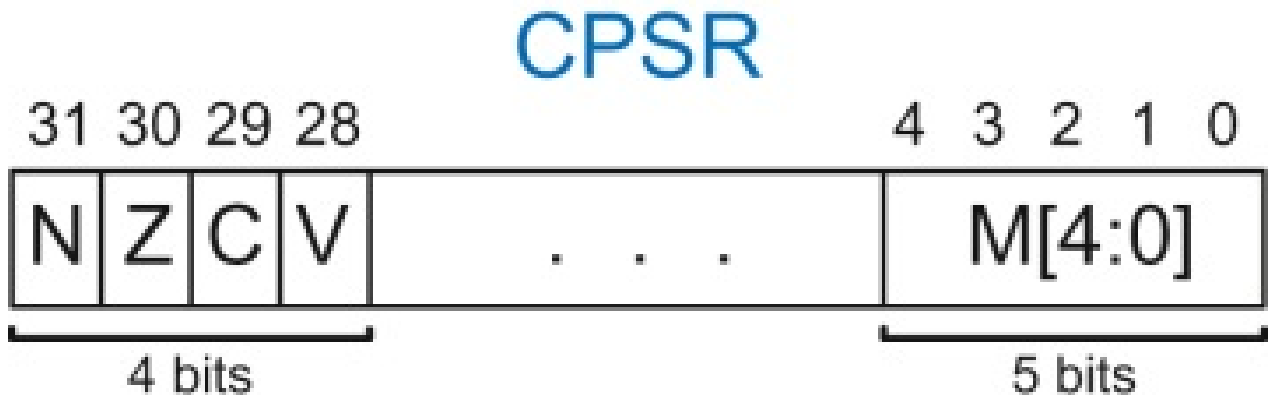
Умножение (Multiplication).

- Има няколко инструкции за умножение в ARM архитектурата
- 32 × 32 умножение, 32 bit резултат
  - MUL R1, R2, R3 – умножава R2 по R3
  - Записва младшите 32 bit в R1, старшите 32 bit се игнорират
- 32 × 32 умножение, 64 bit резултат
  - UMULL R1, R2, R3, R4 - unsigned multiply long
  - Беззнаково умножение на R3 по R4
  - Записва младшите 32 bit в R1, старшите 32 bit – в R2
- 32 × 32 умножение, 64 bit резултат
  - SMULL R1, R2, R3, R4 - signed multiply long
  - Знаково умножение на R3 по R4
  - Записва младшите 32 bit в R1, старшите 32 bit – в R2

КАРХ: Тема_13: ARM архитектура

Флагове (на условията) (Condition Flags).

- Стойността им зависи от резултата на извършваните от процесора операции.
- Съществува набор от условни инструкции, чието изпълнение зависи от текущото състояние на флаговете.
- В ARM архитектурата използваните флагове са *negative* (**N**), *zero* (**Z**), *carry* (**C**) и *overflow* (**V**). Наричат се също и флагове на състоянието (status flags).
- Флаговете се установяват от АЛУ и се записват в старшите 4 бита на 32-bit регистър на състоянието **CPSR** (***C**urrent **P**rogram **S**tatus **R**egister*).



КАРХ: Тема_13: ARM архитектура

Програмни конструкции от високо ниво (high-level software constructs):.

- Разглеждане на:
  - 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_13: ARM архитектура

Преходи (Branching).

- Видове преходи:
  - **Условни (Conditional) – когато условието е изпълнено.** Напр.
    - Преход при равенство (branch if equal) (beq) (Z=1)
    - Преход при неравенсрво (branch if not equal) (bne) (Z=0)
    - Използват се състоянията и на останалите флагове
  - **Безусловни (Unconditional)**
    - branch (B)
    - branch and link (BL)

Преход при равенство (beq).

Пример:

```
# ARM assembly
 MOV R0,#4                  ; R0 = 4
 ADD R1,R0,R0               ; R1 = R0 + R0 = 8
 CMP R0,R1                  ; set flags based on R0-R1=-4, NZCV=1000
 BEQ THERE                  ; branch not taken(Z != 1)
 ORR R1,R1,#1               ; R1 = R1 OR 1 = 9

THERE                       ; етикет (label)
 ADD R1,R1,#78              ; R1 = R1 + 78 = 87
```

Етикетите (Labels) маркират мястото на инструкцията. Те не могат да бъдат запазени думи.

Условно състояние If (If Statement).

High-Level Code

```
if (apples == oranges)
    f = i + 1;
    f = f - i;
```

ARM assembly code

```
; R0 = apples, R1 = oranges
; R2 = f, R3 = i
  CMP R0, R1   ; apples==oranges ?
  ADDEQ R2, R3, #1 ; f=i+1 on Z=1
  SUB R2, R2, R3   ; f=f-i
```

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

- **High-Level Code**

```
if (apples == oranges)
        f = i + 1;
   else
      f = f - i;
```

ARM assembly code

```
; R0 = apples, R1 = oranges
; R2 = f, R3 = i
  CMP R0, R1   ; apples==oranges ?
  ADDEQ R2, R3, #1 ; f=i+1 on Z=1
  SUBNE R2, R2, R3 ; f=f-i on Z=0
```

Цикъл While (While Loops).

- **High-Level 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;
}
```

ARM assembly code

```
; R0 = pow, R1 = x

    MOV R0, #1 ; pow = 1
    MOV R1, #0 ; x = 0

WHILE
    CMP R0, #128 ; pow != 128 ?
    BEQ DONE ; if pow==128 exit
    LSL R0, R0, #1 ; pow = pow*2
    ADD R1, R1, #1 ; x = x + 1
    B WHILE          ; repeat loop
DONE
```

КАРХ: Тема_13: ARM архитектура

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

Цикъл For (For Loops).

- **High-Level Code**

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

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

ARM assembly code

```
; R0 = i, R1 = sum
  MOV R1, #0 ; sum = 0
  MOV R0, #0 ; i = 0
; loop initialization
FOR
  CMP R0, #10 ; i < 10?
  BGE DONE ; if i>=10 exit loop
  ADD R1, R1, R0 ; sum = sum +i
  ADD R0, R0, #1 ; i = i + 1
  B FOR          ; repeat loop
DONE
```

КАРХ: Тема_13: ARM архитектура

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
- ARM архитектурата позволява индексът да се скалира (умножи), да се прибави към базовия адрес и да се извърши зареждане в една инструкция:

LDR R3, [R0, R1, LSL #2] , т.е. R1 се скалира (измества на ляво с 2) и след това се прибавя към базовия адрес (R0), така че полученият адрес от паметта е

R0 + (R1 x 4)

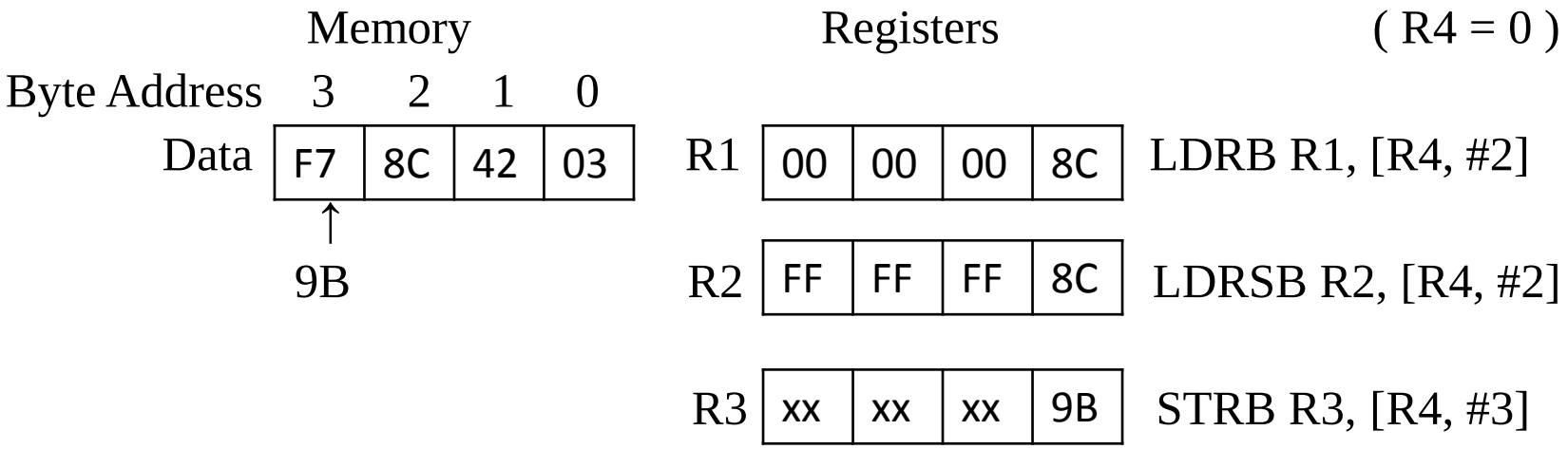
Освен скалирането ARM предлага още *offset*, *pre-indexed* и *post-indexed* адресиране.

Mode	ARM Assembly	Address	Base Register
Offset	LDR R0, [R1, R2]	R1 + R2	Unchanged
Pre-index	LDR R0, [R1, R2]!	R1 + R2	R1=R1+R2
Post-index	LDR R0, [R1], R2	R1	R1=R1+R2

КАРХ: Тема_13: ARM архитектура

Адресиране на отделни байтове от паметта.

- Инструкции *load byte* (LDRB), *load signed byte* (LDRSB) и *store byte* (STRB).
- LDRB запълва старшите байтове с нули (zero-extend).
- LDRSB прави знаково разширение до 32 бита (sign-extend).
- STRB записва най-младшия байт на указания 32 битов регистър на специфициран байтов адрес в паметта.



КАРХ: Тема_13: ARM архитектура

Конвенции при функциите.

- **Caller:**
  - изпраща **arguments** на *callee*
  - Прави обръщение 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_13: ARM архитектура

ARM Конвенции при функциите.

В ARM архитектурата *caller* използва до 4 аргумента и ги слага в регистрите R0÷R3, преди да извика *callee,* която използва за върната стойност регистъра R0 преди да завърши. Използвайки тази конвенция и двете функции знаят къде ще намерят аргументите си и върнатите стойности. *Caller* извиква *callee* използвайки *branch and link* (BL) инструкция. *Callee* **не трябва да променя** памет и регистри използвани от *caller* , т.е. запазващите регистри R4÷R11, LR, стекът и частта от паметта, използвана за временни променливи, остават не променени. Връщането от функцията става с инструкцията MOV PC, LR .

branch and link (BL) инструкцията извършва две операции – запазва адрес за връщане (return address) (на следващата инструкция след BL) в регистъра LR и преход към извиканата функция.

КАРХ: Тема_13: ARM архитектура

Запазени и незапазени регистри.

Запазени	Незапазени
Saved registers : R4÷R11	Temporary register : R12
Stack pointer : SP (R13)	Argument registers : R0÷R3
Return address : LR (R14)	Current Program Status Register
Stack above the stack pointer	Stack below the stack pointer

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

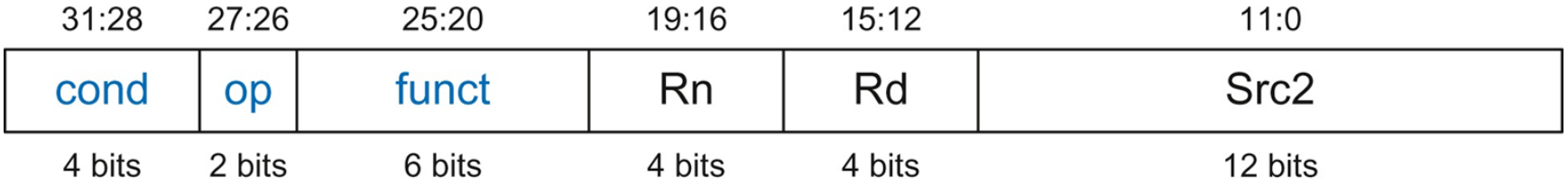
- В ARM архитектурата има три основни формата на инструкциите:
  - *Data-processing*
  - *Memory*
  - *Branch*

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

- **Data-processing** формат е най-общ.
  - Първи операнд източник е регистър (*Rn)* ;
  - Втори операнд източник (*Src2)* – константа или регистър (може отместен);
  - Трети операнд – регистър-цел (*Rd)*.
  - 32 бит инструкция има 6 полета: *cond*, *op*, *funct*, *Rn*, *Rd*, *Src2*

Data-processing



op – opcode , (= 00 за Data-processing)

cond – conditional execution в зависимост от състоянието на флаговете

(за инструкции без условие *cond* = 1110_2)

funct – function code, съдържа 3 подполета – *I*, *cmd* и *S*.

I=1, когато *Src2 е константа*. *S*=1, когато инструкцията установява флаг.

cmd – указва специфична команда , напр. ADD, SUB, …

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

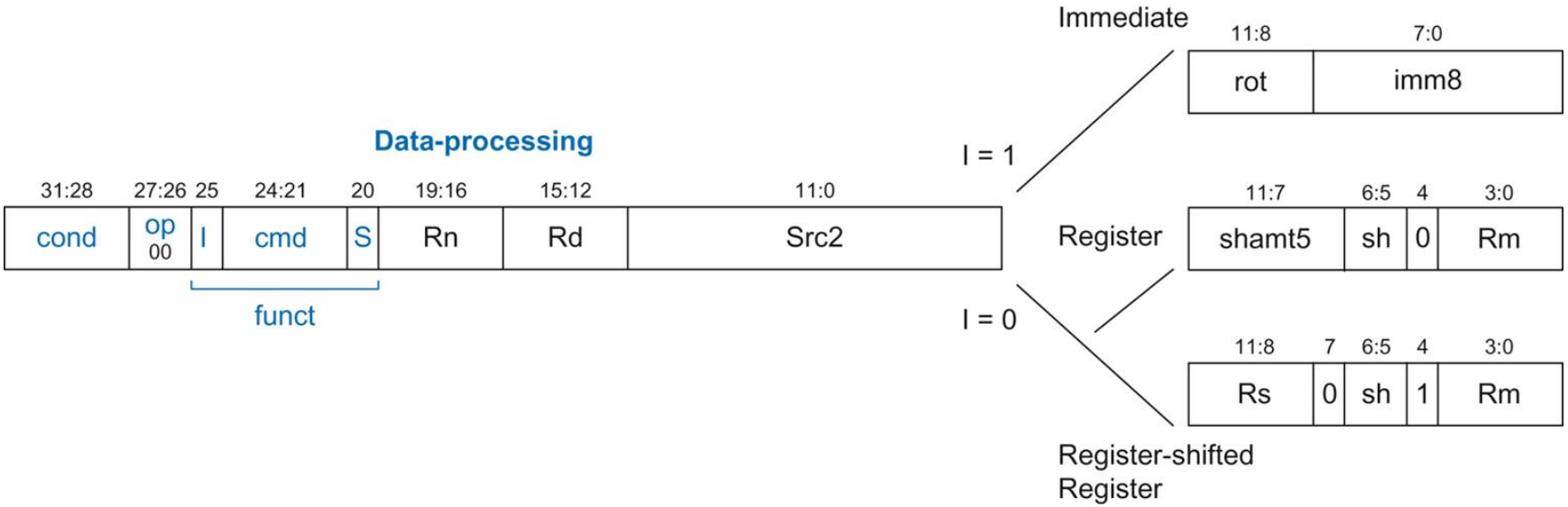
- **Data-processing**

– Три формата за декодиране на *Src2* :

1) константа;

2) регистър (Rm), възможно отместен с константа (shamt5);

3) регистър (Rm), отместен от друг регистър (Rs), sh – определя вида на местенето



– *imm8* – 8-бит константа, която с *2 x rot* се допълва до 32 бит.

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

- **Data-processing**

– Пример – с три регистъра.

Assembly Code

```
ADD R5, R6, R7
    (0xE0865007)
SUB R8, R9, R10
    (0xE049800A)
```

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
1110_2	00_2	0	0010_2	0	9	8	0	0	0	10
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm

Machine Code

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110	00	0	0100	0	0110	0101	00000	00	0	0111
1110	00	0	0010	0	1001	1000	00000	00	0	1010
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm

– Пример – с два регистъра и константа.

Assembly Code

```
ADD R0, R1, #42
    (0xE281002A)
SUB R2, R3, #0xFF0
    (0xE2432EFF)
```

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
1110_2	00_2	1	0010_2	0	3	2	14	255
cond	op	I	cmd	S	Rn	Rd	rot	imm8

Machine Code

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110	00	1	0100	0	0001	0000	0000	00101010
1110	00	1	0010	0	0011	0010	1110	11111111
cond	op	I	cmd	S	Rn	Rd	rot	imm8

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

- **Data-processing**

– Иструкции с местене (shift).

Декодиране на *sh* :

LSL – 00_2

LSR – 01_2

ASR – 10_2

ROR – 11_2

– Пример – с директно местене.

Assembly Code

```
LSL  R0, R9, #7
     (0xE1A00389)
ROR  R3, R5, #21
     (0xE1A03AE5)
```

Field Values

	31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
	cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
LSL	1110_2	00_2	0	1101_2	0	0	0	7	00_2	0	9
ROR	1110_2	00_2	0	1101_2	0	0	3	21	11_2	0	5

Machine Code

	31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
	cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
LSL	1110	00	0	1101	0	0000	0000	00111	00	0	1001
ROR	1110	00	0	1101	0	0000	0011	10101	11	0	0101

– Пример – с регистърно местене.

Assembly Code

```
LSR R4, R8, R6
    (0xE1A04638)
ASR R5, R1, R12
    (0xE1A05C51)
```

Field Values

	31:28	27:26	25	24:21	20	19:16	15:12	11:8	7	6:5	4	3:0
	cond	op	I	cmd	S	Rn	Rd	Rs		sh		Rm
LSR	1110_2	00_2	0	1101_2	0	0	4	6	0	01_2	1	8
ASR	1110_2	00_2	0	1101_2	0	0	5	12	0	10_2	1	1

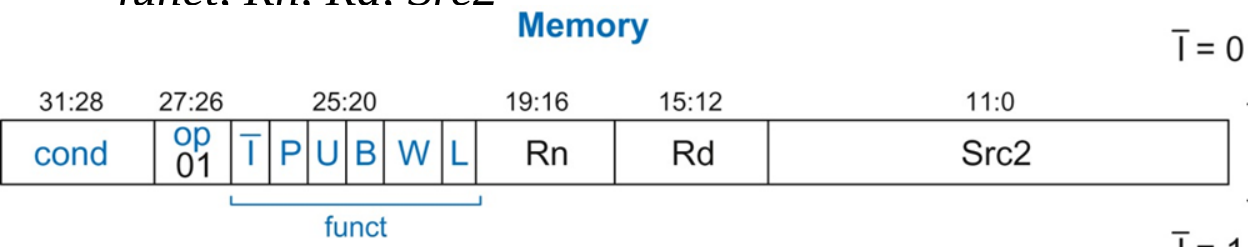
Machine Code

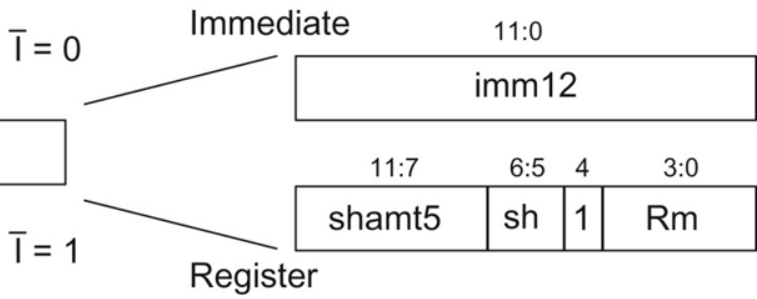
	31:28	27:26	25	24:21	20	19:16	15:12	11:8	7	6:5	4	3:0
	cond	op	I	cmd	S	Rn	Rd	Rs		sh		Rm
LSR	1110	00	0	1101	0	0000	0100	0110	0	01	1	1000
ASR	1110	00	0	1101	0	0000	0101	1100	0	10	1	0001

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

- **Memory**
  - 32 бит инструкция има 6 полета: *cond*, *op*, *funct*. *Rn*. *Rd*. *Src2*





L	B	Instruction
0	0	STR
0	1	STRB
1	0	LDR
1	1	LDRB

P	W	Index Mode
0	0	Post-Index
0	1	Not supported
1	0	Offset
1	1	Pre-Index

  - *op* = 01
  - *Rn* – базов регистър
  - *Src2* – офсет
  - *Rd* – load/store регистър

Bit	I	U
0	Immediate offset in Scr2	Subtract offset from
1	Register offset in Scr2	Add offset to base

КАРХ: Тема_13: ARM архитектура

Формат на инструкциите в ARM архитектурата.

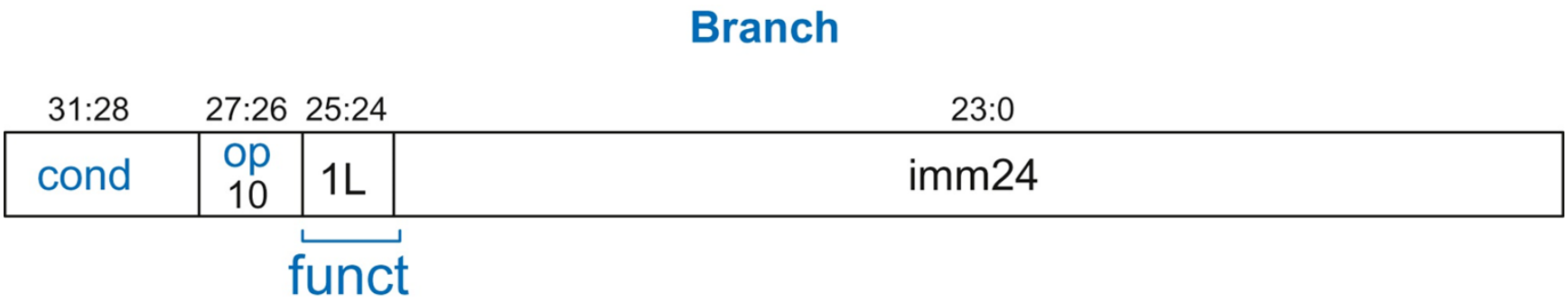
- **Branch**

– 32 бит инструкция има 4 полета: *cond*, *op*, *funct*, *Imm24*

– *op* = 10

– *funct* =1L , L=1 (BL) , L=0 (B)

– *Imm24* – 24 bit константа със знак, задава адреса на следващата инструкция, относително PC+8



КАРХ: Тема_13: ARM архитектура

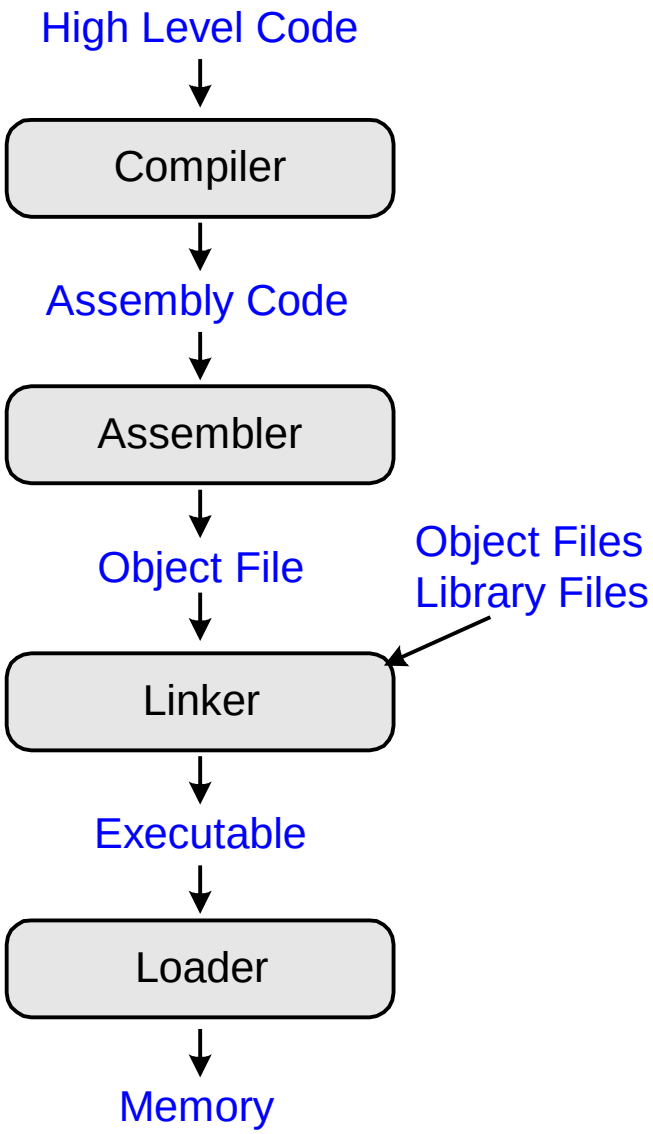
Методи на адресация (Addressing Modes).

Как се адресират операндите?

- Регистърно
  - Register Only — ADD R3, R2, R1 — R3 ← R2 + R1
  - Immediate-shifted register — SUB R4, R5, R9, LSR #2 — R4 ← R5 – (R9 >> 2)
  - Register-shifted register — ORR R0, R9, R2, ROR R7 — R0 ← R9 | (R2 ROR R7)
- Непостедствено (Immediate) — SUB R3, R2, #25 — R3 ← R2 – 25
- Чрез базов адрес (Base Addressing)
  - Immediate offset — STR R6, [R11, #77] — mem[R11+77] ← R6
  - Register offset — LDR R12, [R1, – R5] — R12 ← mem[R1 – R5]
  - Immediate-shifted register offset — LDR R8, [R9, R2, LSL #2] — R8 ← mem[R9+(R2<<2)]
- Относително програмния брояч (PC-Relative) — B LABEL — Branch to LABEL

КАРХ: Тема_13: ARM архитектура

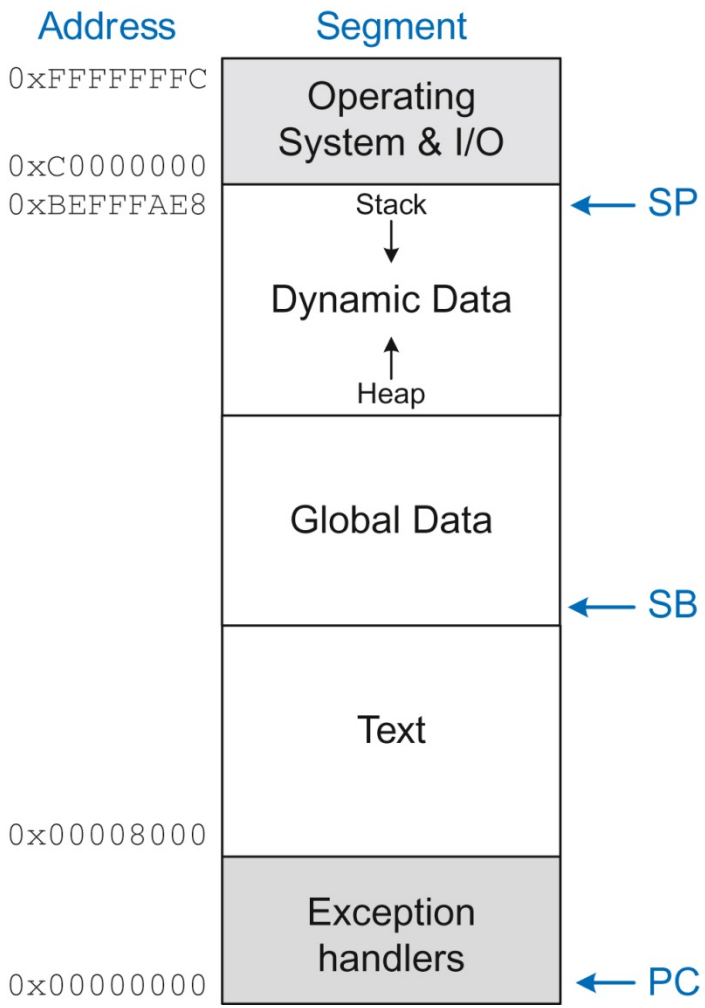
Компилиране и стартиране на програмите.



КАРХ: Тема_13: ARM архитектура

Компилиране и стартиране на програмите.

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



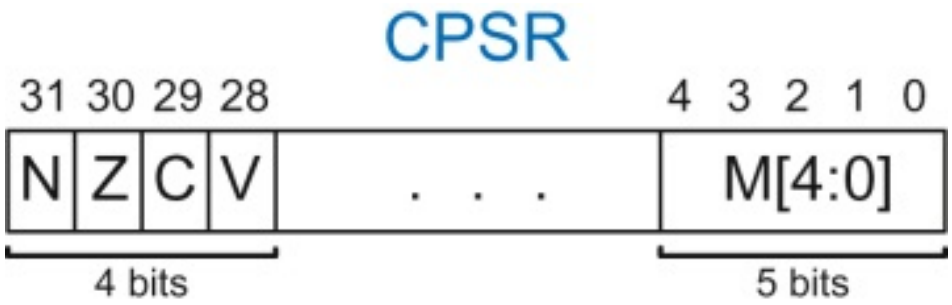
КАРХ: Тема_13: ARM архитектура

Прекъсвания (Exceptions)

- Нередовна (невалидна) операция предизвиква преход към програма за обработка на прекъсванията (*exception handler*). Работата на тази програма зависи от това в какъв мод е изпълняваната програма (различно ниво на привилегии).

В ARM архитектурата има няколко мода (ARM execution modes). Какъв е модът на текущо изпълняваната програма се разбира от регистъра на състоянието **CPSR** (***C**urrent **P**rogram **S**tatus **R**egister*).

Mode	$CPSR_{4:0}$
User	10000
Supervisor	10011
Abort	10111
Undefined	11011
Interrupt (IRQ)	10010
Fast Interrupt (FIQ)	00000



КАРХ: Тема_13: ARM архитектура

Еволюция на ARM архитектурата.

- ARM1 процесор произведен във Великобритания за BBC от Acorn Computer през 1985 година като ъпгрейд на процесора 6502.
- През 1986 г. – ARM2 влиза в производство отAcorn Archimedus Computer. ARM е акроним на Acorn RISC Machine. Използва се Версия 2 (ARMv2) на набора инструкции. Адресната шина е 26 bit, (6 bit – status bits). Скоро адресирането е увеличено до 32 bit, а status bits – преместени в **CPSR.**
- През 1993 излиза ARMv4 , (която разгледахме).
- ARMv5TE добавя DSP инструкции.
- ARMv6 добавя мултимедийни инструкции.
- ARMv7 – с подобрени мултимедийни инструкции и инструкции с плаваща запетая.
- ARMv8 – напълно нова 64 bit архитектура.