

ARM zbirnik – OR nadgradnja

*Arhitektura in
programiranje v zbirniku*

Spletni simulator cpulator

- CPUlator ARMv7 System Simulator (01xz.net)

Registers

Register	Value
r0	00000028
r1	00000040
r2	00000010
r3	00000050
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000048

Editor (Ctrl-E)

```
1 .text
2 .org 0x20
3 @spremenljivke
4 stev1: .word 0x40
5 stev2: .word 0x10
6 rez: .space 4
7
8 .align
9 .global _start
10 _start:
11
12 @program
13 adr r0, stev1
14 ldr r1, [r0]
15
16 adr r0, stev2
17 ldr r2, [r0]
18
19 add r3, r2, r1
20
21 adr r0, rez
22 str r3, [r0]
23
24 end: b end
```

Memory (Ctrl-M)

Address	Memory contents and ASCII
00000040	20 00 4f e2 00 30 80 e5
00000050	aa aa aa aa aa aa aa aa
00000060	aa aa aa aa aa aa aa aa
00000070	aa aa aa aa aa aa aa aa
00000080	aa aa aa aa aa aa aa aa
00000090	aa aa aa aa aa aa aa aa
000000a0	aa aa aa aa aa aa aa aa
000000b0	aa aa aa aa aa aa aa aa
000000c0	aa aa aa aa aa aa aa aa
000000d0	aa aa aa aa aa aa aa aa
000000e0	aa aa aa aa aa aa aa aa
000000f0	aa aa aa aa aa aa aa aa
00000100	aa aa aa aa aa aa aa aa
00000110	aa aa aa aa aa aa aa aa
00000120	aa aa aa aa aa aa aa aa
00000130	aa aa aa aa aa aa aa aa
00000140	aa aa aa aa aa aa aa aa
00000150	aa aa aa aa aa aa aa aa
00000160	aa aa aa aa aa aa aa aa
00000170	aa aa aa aa aa aa aa aa
00000180	aa aa aa aa aa aa aa aa
00000190	aa aa aa aa aa aa aa aa
000001a0	aa aa aa aa aa aa aa aa
000001b0	aa aa aa aa aa aa aa aa
000001c0	aa aa aa aa aa aa aa aa
000001d0	aa aa aa aa aa aa aa aa
000001e0	aa aa aa aa aa aa aa aa

Messages

```
Compiling...
Code and data loaded from ELF executable into memory. Total size is 80 bytes.
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmhSiYoH.s.o work/asmhSiYoH.s
Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmhSiYoH.s.elf work/asmhSiYoH.s.o
Compile succeeded.
```

Začetni projekt OR

Pripomočki

▼ Laboratorijske vaje ↗

ARM Ref. in OR QuickRef

CPUlator ARMv7 System Simulator (01xz.net)

ARMv4T Partia

Operation		Syntax	
Move	Move	<code>mov{cond}[s] Rd, shift_op</code>	1/2
	with NOT	<code>mvn{cond}[s] Rd, shift_op</code>	1/2
	CPSR to register	<code>mrs{cond} Rd, cpsr</code>	1/2
	SPSR to register	<code>mrs{cond} Rd, spsr</code>	1/2
	register to CPSR	<code>msr{cond} cpsr_fields, Rm</code>	1/2
	register to SPSR	<code>msr{cond} spsr_fields, Rm</code>	1/2
	immediate to CPSR	<code>msr{cond} cpsr_fields, #immsr</code>	1/2
immediate to SPSR	<code>msr{cond} spsr_fields, #immsr</code>	1/2	

Zbirnik, Assembler

- DATOTEKA
Seznam ukazov zbornika ARM ↗
- DATOTEKA
FRI ARM Zbirnik Quickref A4 v0.4 ↗

ARM zbirnik Quick Reference (v0.4)

(Pripomoček za izvedbo laboratorijskih vaj pri predmetu Organizacija računalnikov)

Načini naslavljanja:

Bazno naslavljanje: $A = r0 + D$ (odmik)
 D .. dolžina krajša od dolžine naslova

Indeksno naslavljanje:
 Kadar je odmik D enak dolžini naslova
 Lahko ga nadomestimo z $D1 = r1 + D$ in dobimo :
 $A = r0 + r1 + D = r0 + D1$

Fovzetek načinov naslavljanja		
Način naslavljanja	Primer	ODM je lahko:
Posredno nasl.	<code>ldr r1, [r0,ODM]</code>	• brez » «
Posr. s pred-ind.	<code>ldr r1, [r0,ODM]!</code>	• #odmik »#-4«
Posr. s po-ind.	<code>ldr r0, [r1],ODM</code>	• register »r1 «
		• reg. s pom. »r2,LSL #2«

1. Posredno (bazno) naslavljanje brez odmika

```
adr r0, stevi
ldr r1, [r0] @ r1 <- mem32[r0]
adr r1, stevi @ r1 <- mem32[r0]
```

11. Avtomatsko po-indeksiranje

```
ldr r0, [r1], #2 ; r0 <- me
```

12. Avtomatsko po-indeksiranje odmikom:

```
ldr r0, [r1], #2, LSL #2 ;
```

Razširitev ničle / razširitev predznaka

Pri nalaganju 8 in 16-bitnih podatkov potrebno razširiti predznak al operacije 32 bitni.

- pri nepredznačenih ope `ldr, ldrb`
- pri predznačenih opera `ldrb, ldrrb`

Primerjave nepredznačenih/predznačenih

Video posnetki

Vse skupine

LAB Objave Datoteke Zapiski +

+ Novo Naloži Uredi v mrežnem po

LAB

OR VSP 2023/24

Domača stran

Zvezek za predavanja

Classwork

Dodeljene naloge

Ocene

Reflect

Insights

Kanali

Splošno

LAB

Predavanja

Vprašanja in odgovori

Ime

OR LAB 01.1 Ponovitev znanja zbornika RA...

OR LAB 01.2 Naloga 1.1.mp4

OR LAB 01.3 Naloga 1.2.mp4

OR LAB 01.4 Naloga 1.3.mp4

Začetni projekt OR

Logični ukazi (delo z določenimi biti)

and r1, r2, r3 @brisanje z ničlo v maski določenih bitov

r2	00000000	00000000	00000000	01010011
and r3	11111111	11111111	11111111	11001010
=r1	00000000	00000000	00000000	01000010

bic r1, r2, r3 @brisanje z enico v maski določenih bitov

r2	00000000	00000000	00000000	01010011
bic r3	11111111	11111111	11111111	11001010
=r1	00000000	00000000	00000000	00010001

orr r1, r2, r3 @postavljanje z enico v maski določenih bitov

r2	00000000	00000000	00000000	01010011
or r3	11111111	11111111	11111111	11001010
=r1	11111111	11111111	11111111	11011011

eor r1, r2, r3 @invertiranje z enico v maski določenih bitov

r2	00000000	00000000	00000000	01010011
eor r3	11111111	11111111	11111111	11001010
=r1	11111111	11111111	11111111	10011001

Logični ukazi (preverjanje stanja določenih bitov)

- Preverjanje stanja enega bita (določen je z enico v maski)

`tst r1, r2` @zastavice postavi glede na `r1 AND r2`

r1	00000000	00000000	00000000	01010011
tst r2	00000000	00000000	00000000	00000100
=	00000000	00000000	00000000	00000000

 → z=1

r1	00000000	00000000	00000000	01010011
tst r2	00000000	00000000	00000000	00000110
=	00000000	00000000	00000000	00000010

 → z=0

- Preverjanje stanja večih bitov:

- Najprej izločimo bite, ki nas zanimajo: `and`
- Primerjamo z želenim stanjem bitov (bite, ki nas ne zanimajo, primerjamo z 0)

Zgled:

@preveri, da je bit7 v r1 enak 0 in bit 2 v r1 enak 1

`and r2, r1, #0x84` @0x84 = 00...010000100 => r2 = 00..00?0000?00

`cmp r2, #0x04` @0x04 = 00...000000100; ustreza, če Z=1

Aritmetično-logični ukazi, seznam

• Aritmetični ukazi:

```
add r0, r1, r2      ; r0 <- r1 + r2
adc r0, r1, r2      ; r0 <- r1 + r2 + C      (add with C)
sub r0, r1, r2      ; r0 <- r1 - r2
sbc r0, r1, r2      ; r0 <- r1 - r2 + C - 1  (-not(C)=- (1-C)= C-1
rsb r0, r1, r2      ; r0 <- r2 - r1          (reverse subtract)
rsc r0, r1, r2      ; r0 <- r2 - r1 + C - 1  (rev. sub -not(C) )
```

• Logični ukazi:

```
and r0, r1, r2      ; r0 <- r1 AND r2
orr r0, r1, r2      ; r0 <- r1 OR r2
eor r0, r1, r2      ; r0 <- r1 XOR r2
bic r0, r1, r2      ; r0 <- r1 AND NOT r2
```

• Prenos med registri:

```
mov r0, r2          ; r0 <- r2
mvn r0, r2          ; r0 <- NOT r2
```

• Primerjave:

```
cmp r1, r2          ; set CPSR flags on r1 - r2
cmn r1, r2          ; set CPSR flags on r1 + r2
tst r1, r2          ; set CPSR flags on r1 AND r2
teq r1, r2          ; set CPSR flags on r1 XOR r2      (equivalence test)
```

LAB - 1 : Tabla

DELO Z BITI

BRISANJE BITOV:

AND (IN)

OPERAND	X	X
MASKA	0	1
REZULTAT	0	X

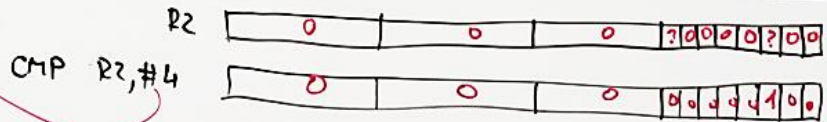
BRISAJE BITE, KIER JE MASKA 0
 PUSTI ——— "1" ——— 1

BIC (Bit Clear)

OPERAND	X	X
MASKA	0	1
REZULTAT	X	0

BRISAJE BITE, KIER JE MASKA 1
 PUSTI ——— "1" ——— 0

bit₇ = 0 in bit₂ = 1 v R7?



POSTAVLJANJE BITOV:

ORR (ALI)

OPERAND	X	X
MASKA	1	0
REZULTAT	1	X

POSTAVI BITE, KIER JE MASKA 1
 PUSTI ——— "1" ——— 0

NEGIRANJE INVERTIRANJE BITOV:

EOR (EKSKL. ALI)

OPERAND	X	X
MASKA	1	0
REZULTAT	X	X

NEGIRA BITE, KIER JE MASKA 1
 PUSTI ——— "1" ——— 0

TESTIRANJE BITOV:

TST

"AND": BREZ REZULTATA
 . VPLIVA NA ZASTAVICE

LAHKO PREVERIMO:

- STANJE POLJUDNEGA BITA (= 0, ≠ 0)
 - MASKA BITA JE 1
 - MASKA OSTALIM JE 0
- STANJE VEČIH BITOV (= 0, ≠ 0)
 - MASKA BITOV JE 1
 - MASKA OSTALIM JE 0

AND + CMP

- LAHKO PREVERIMO STANJE VEČIH BITOV NA POLJUDNO VREDNOST

ZKORAKI:

- ① AND: "MASKIRANO" VSE OSTALE BITE
- ② CMP: POREJANO Z ŽELEVNIM STANJETI RBDANIM BITOV