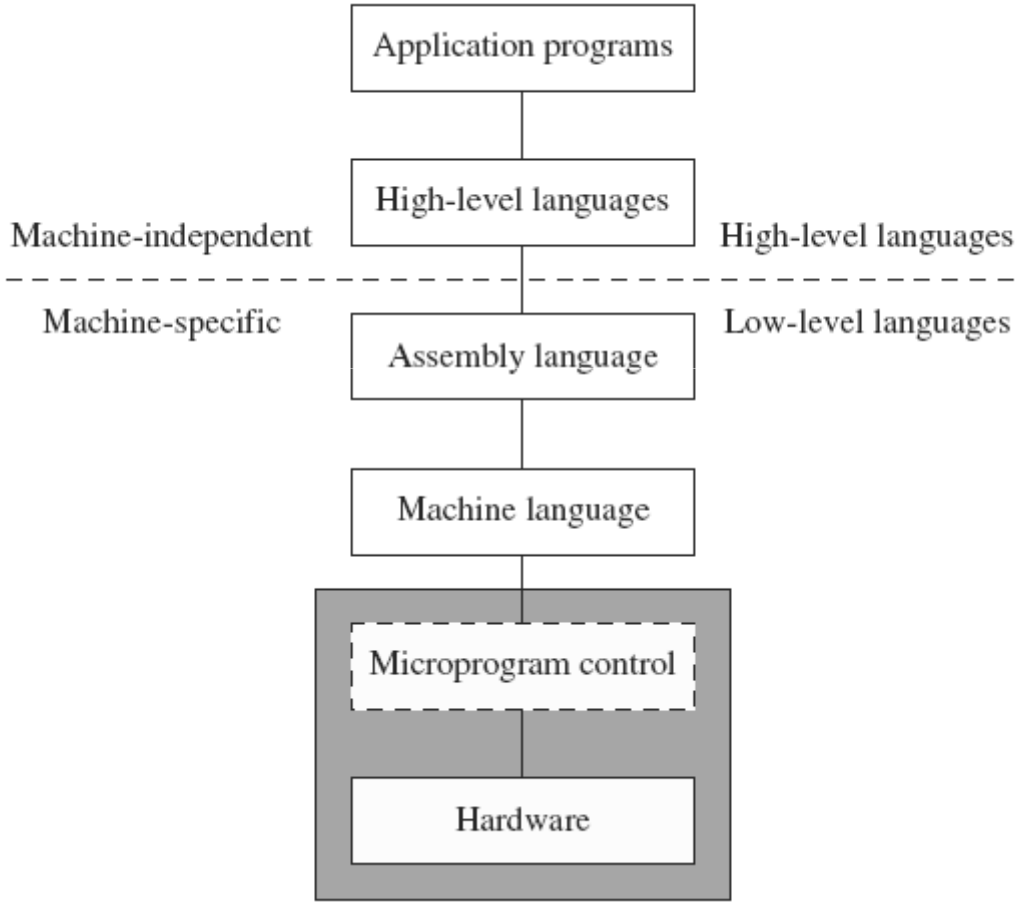
Arhitectura Embedded

Curs 2

ISA

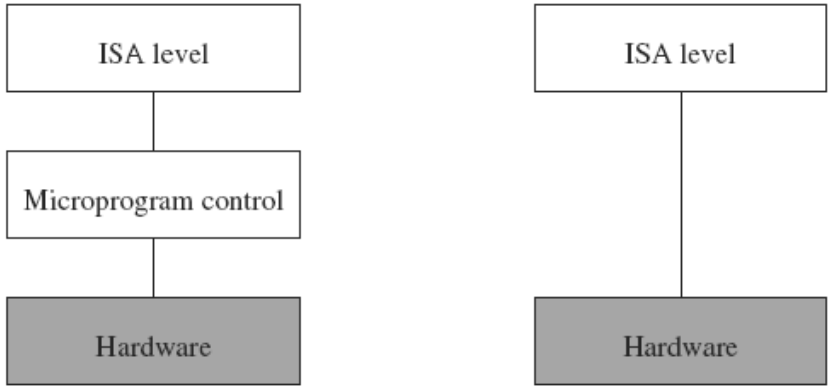
- Folosim arhitectura setului de instructiuni (ISA) pentru a abstractiza functionarea interna a unui procesor.
- ISA defineste “personalitatea” unui procesor
  - Cum functioneaza procesorul
  - Ce fel de instructiuni executa
  - Care este interpretarea data instructiunilor
- Folosind ISA bine definit, putem descrie un procesor la nivel logic
- Chipuri cu structura hardware total diferita pot folosi aceeasi ISA
  - Ex: Intel Pentium, Celeron si Xeon folosesc IA-32

Sistem de calcul – programmer's view



Processor Design

- La ora actuala exista doua filozofii de design pentru un procesor
  - Complex Instruction Set Computer (CISC)
  - Reduced Instruction Set Computer (RISC)
- Cum se pot deosebi din perspectiva ISA?



(a) CISC implementation (b) RISC implementation

Numarul de adrese pe instructiune

- O caracteristica ISA ce defineste arhitectura procesorului o constituie numarul de adrese folosite intr-o instructiune
- Majoritatea operatiilor sunt de doua tipuri:
  - Unare (not, and, xor etc.)
  - Binare (add, sub, mul …)
- Majoritatea procesoarelor folosesc trei adrese: operanzi si destinatie
  - add(dest, op1, op2) -> dest = op1+op2
  - mul(dest, op1, op2) -> dest = op1*op2
- Putem reduce numarul adreselor la doua
  - add(op1,op2) -> op1=op1+op2
- Sau chiar la una…sau zero adrese.

Three address machines

- Instructiunea contine toate cele trei adrese
- Exemplu:

$$A = B + C * D - E + F + A$$

mult T,C,D ;	T = C*D	4
add T,T,B ;	T = B + C*D	4
sub T,T,E ;	T = B + C*D – E	4
add T,T,F ;	T = B + C*D - E + F	4
add A,T,A ;	A = B + C*D - E + F + A	4

Memory access

Two address machines

- In exemplul anterior, aproape toate instructiunile foloseau de doua ori aceeasi adresa.
  - De ce sa nu folosim o acea adresa pe post de sursa si destinatie?
- IA-32 foloseste aceasta schema

$$A = B + C * D - E + F + A$$

load T,C ;	T = C	3
mult T,D ;	T = C*D	4
add T,B ;	T = B + C*D	4
sub T,E ;	T = B + C*D - E	4
add T,F ;	T = B + C*D - E + F	4
add A,T ;	A = B + C*D - E + F + A	4

Memory access

One address machines

- Folosesc un registru special pentru a incarca operandul si a stoca rezultatul (registru acumulator)
- Au fost printre primele arhitecturi de procesoare
  - Necesita cel mai mic spatiu de memorie dintre toate arhitecturile

$$A = B + C * D - E + F + A$$

```
load C ;      load C into the accumulator
mult D ;        accumulator = C*D
add B ;         accumulator = C*D+B
sub E ;         accumulator = C*D+B-E
add F ;        accumulator = C*D+B-E+F
add A ;      accumulator = C*D+B-E+F+A
store A ; store the accumulator contents in A
```

Memory access
2
2
2
2
2
2
2

Zero address machines

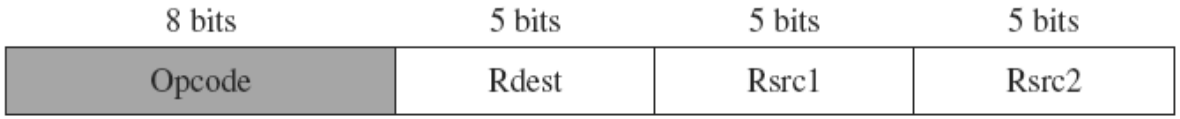
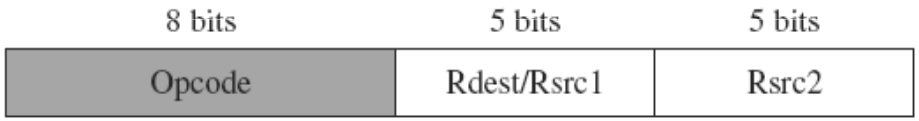
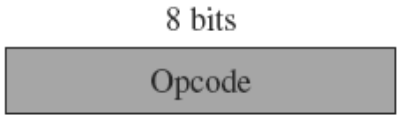
- Folosesc stiva pentru stocarea operanzilor si a rezultatului – stack machines

A = B + C * D – E + F + A

push E ;	<E>	2
push C ;	<C, E>	2
push D ;	<D, C, E>	2
mult ;	<C*D, E>	1
push B ;	<B, C*D, E>	2
add ;	<B+C*D, E>	1
sub ;	<B+C*D-E>	1
push F ;	<F, B+D*C-E>	2
add ;	<F+B+D*C-E>	1
push A ;	<A, F+B+D*C-E>	2
add ;	<A+F+B+D*C-E>	1
pop A ;	< >	2

Memory access

Comparatie

Formatul instructiunii	Len	Accese la memorie
 3-address format	23	20
 2-address format	18	23
 1-address format	13	14
 0-address format	8	19

CISC World

- Arhitectura dominanta pe piata PC-urilor este cea Intel.
- Procesoarele folosesc un set complex de instructiuni (CISC).
- Care este motivatia actuala?
  - Inchiderea prapastiei semantice (semantic gap).
- Efectele secundare ale acestei decizii sunt greu de ignorat.

RISC vs. CISC

- De ce RISC e mai bun decat CISC in embedded (si nu numai) ?

  La inceputul erei calculatoarelor, procesoarele aveau un design simplu si un numar limitat de functii

  In timp, functiile au devenit tot mai numeroase

  – Adaugarea de instructiuni complexe a dus la complicarea design-ului procesorului
  – Mai multe tranzistoare pe chip:
    - costuri ridicate de productie
    - consum foarte mare de energie
    - probleme cu disiparea caldurii

RISC vs. CISC

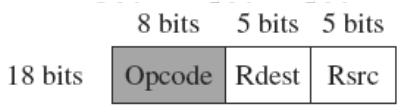
Regula 80-20 : 80% din timp se folosesc 20% din instructiunile unui procesor.

1. Multe instructiuni CISC sunt nefolosite de catre programatori
2. Majoritatea instructiunilor complexe pot fi “sparte” in mai multe instructiuni simple

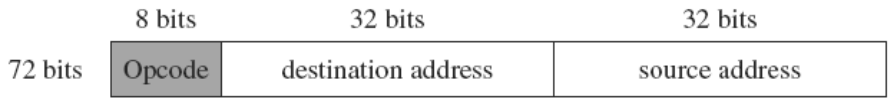
RISC vs. CISC

- Exemplu. Inmultirea a doua numere:

RISC	CISC
LOAD R1, D1 LOAD R2, D2 PROD R1, R2 STORE D1, R1	MULT D1, D2

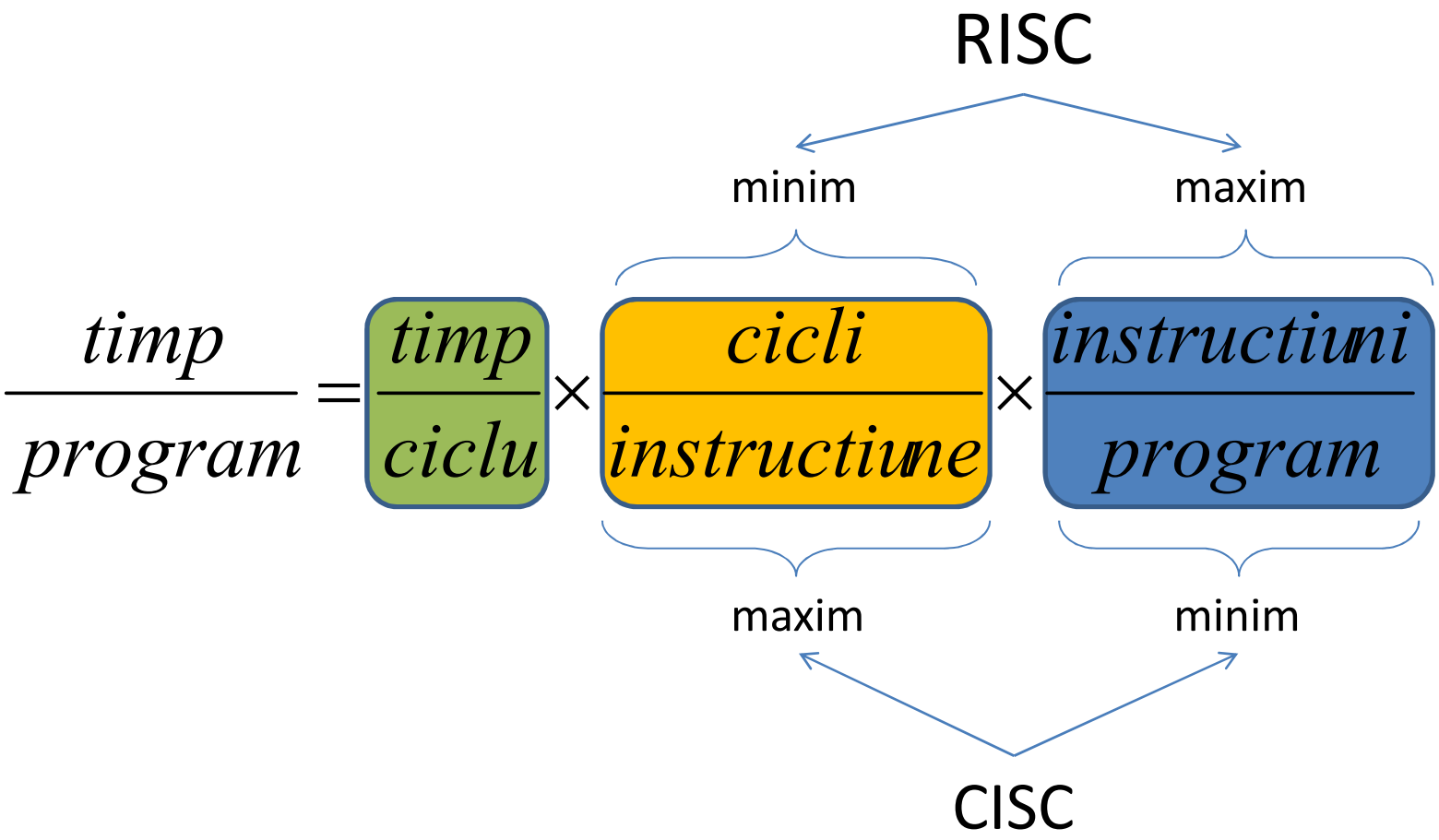


1. Arhitectura Load & Store
2. Fiecare instructiune se executa intr-un singur ciclu de ceas
3. Dupa executie registrele R1 si R2 raman initializate
4. Mai multe linii de cod -> memorie de program mai mare



1. Load/Store se executa transparent
2. Instructiune complexa – poate sa dureze mai multe cicluri de ceas
3. Dupa executie registrele generale sunt aduse la zero
4. O singura linie de cod

Timpul de Executie



RISC vs. CISC: Concluzie

RISC	CISC
1. Pune accent pe software 2. Instructiuni reduse intr-un singur ciclu de ceas 3. Load & Store ca instructiuni separate 4. Cod cu multe instructiuni 5. Complexitate redusa -numar mic de tranzistoare pe chip (lasa loc de periferice) 6. Necesita un spatiu marit de memorie pentru program si date	1. Accent pe hardware 2. Instructinui complexe in unul sau mai multi cicli 3. Load/Store incorporate in instructiunea complexa 4. Cod de lungime redusa 5. Complexitate marita – numar mare de tranzistoare alocate executarii instructiunilor complexe 6. Nu are nevoie de foarte multa memorie

RISC vs. CISC

- In trecut piata era dominata de procesoare CISC
- Ce a blocat dezvoltarea RISC?
  - Tehnologiile existente in trecut
  - Lipsa software (compilatoare doar pentru x86)
  - Pretul ridicat al memoriilor
  - Lipsa de interes a pietei
  - Intel

RISC vs. CISC

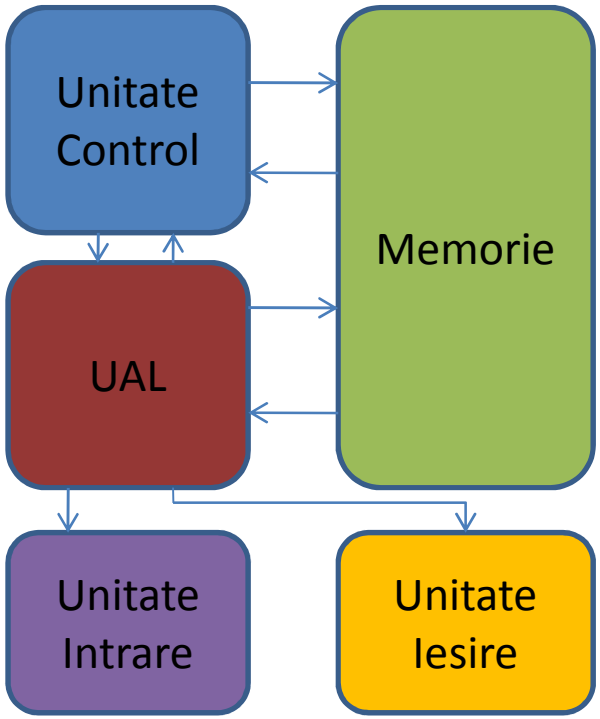
- Ce se intampla in prezent
  - Granita dintre cele doua arhitecturi este neclara
  - Majoritatea procesoarelor au o arhitectura hibrida
    - Intel Pentium: interpretor CISC peste un nucleu RISC
  - Procesoarele CISC nu sunt potrivite pentru embedded
- RISC a monopolizat domeniul Embedded
  - Cost redus al memoriei
  - Compilatoare eficiente
  - Costuri mici de fabricatie
  - Consum redus de energie
  - Eficienta crescuta

RISC si Embedded

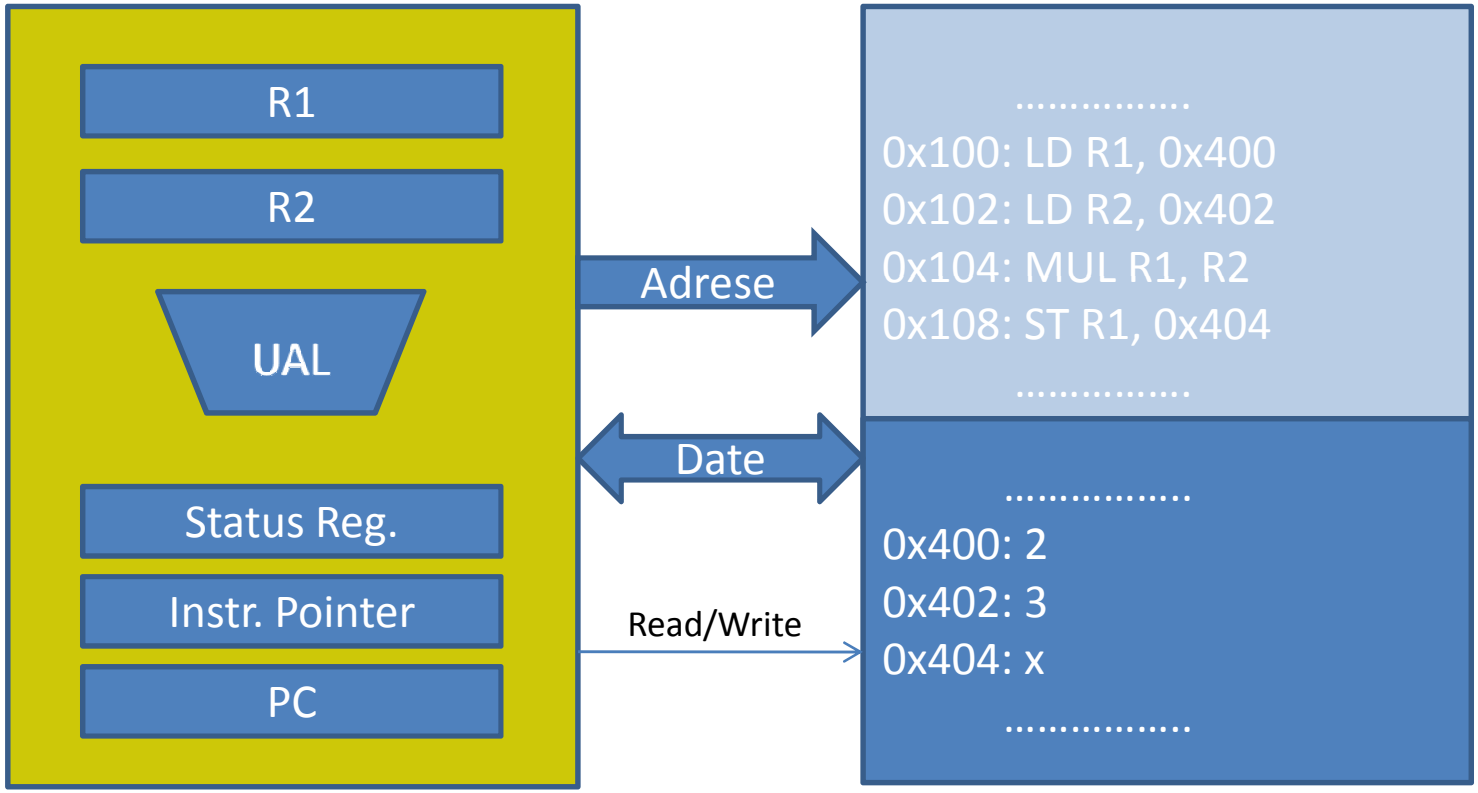
- Procesoare RISC
  - ARM
  - MIPS
  - Power Architecture (IBM, Freescale)
  - Sun SPARC
  - Atmel AVR
  - Microchip PIC

Arhitectura von Neumann

- Consta dintr-o unitate de procesare si o singura memorie.
- In memorie coexista date si instructiuni.
- Sunt masini care pot sa-si modifice singure programul.

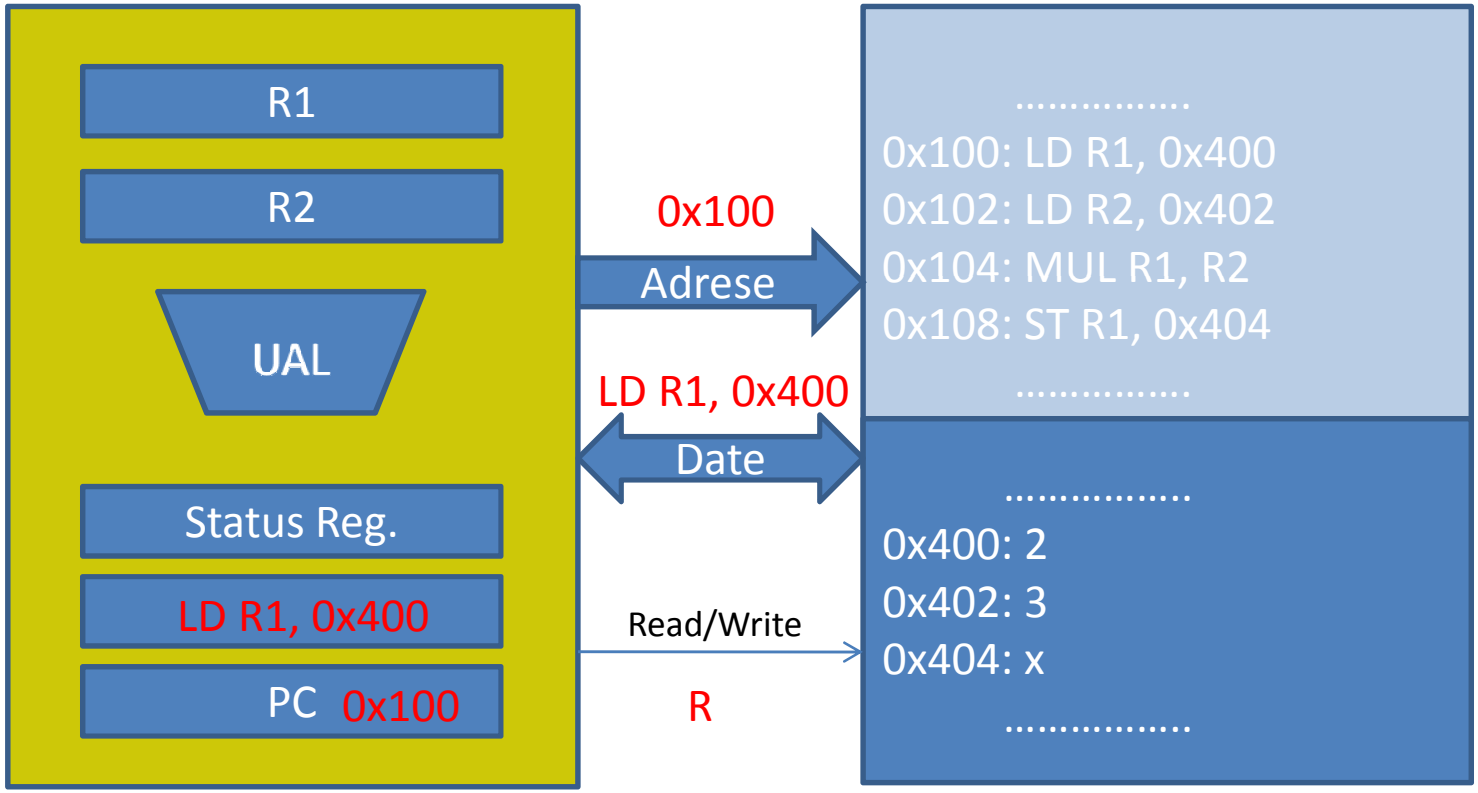


Arhitectura von Neumann



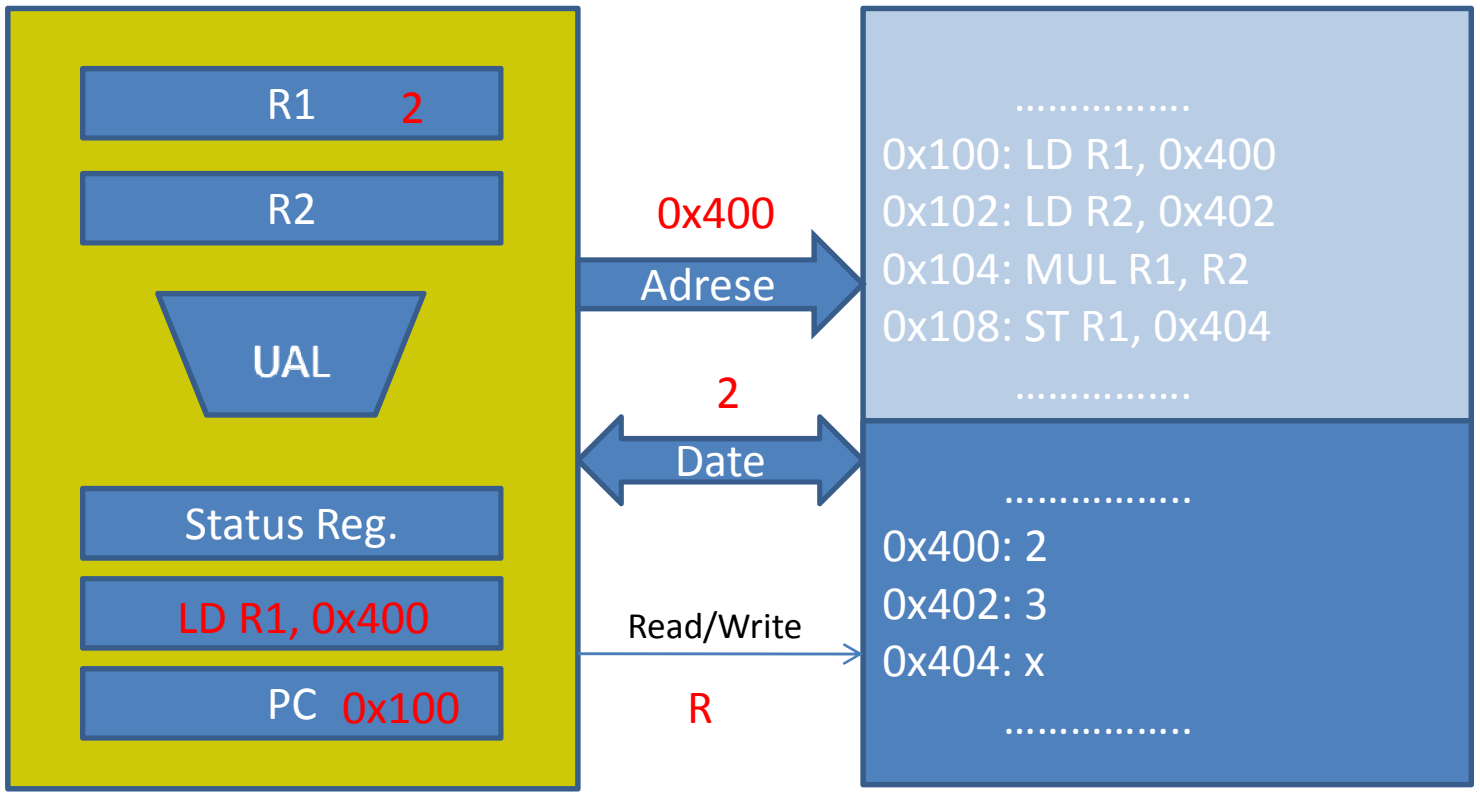
Arhitectura von Neumann

Fetch 0x100



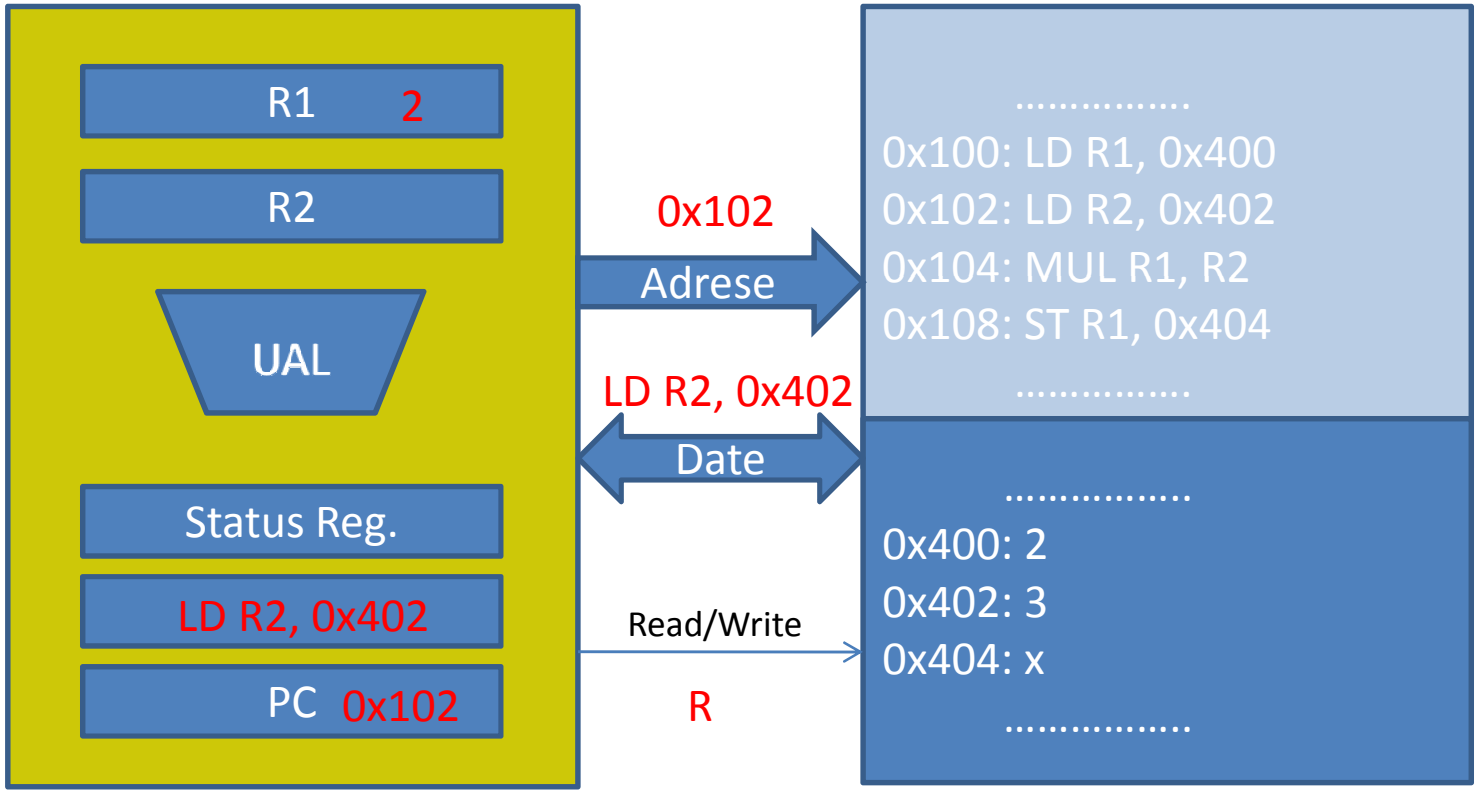
Arhitectura von Neumann

Execute 0x100



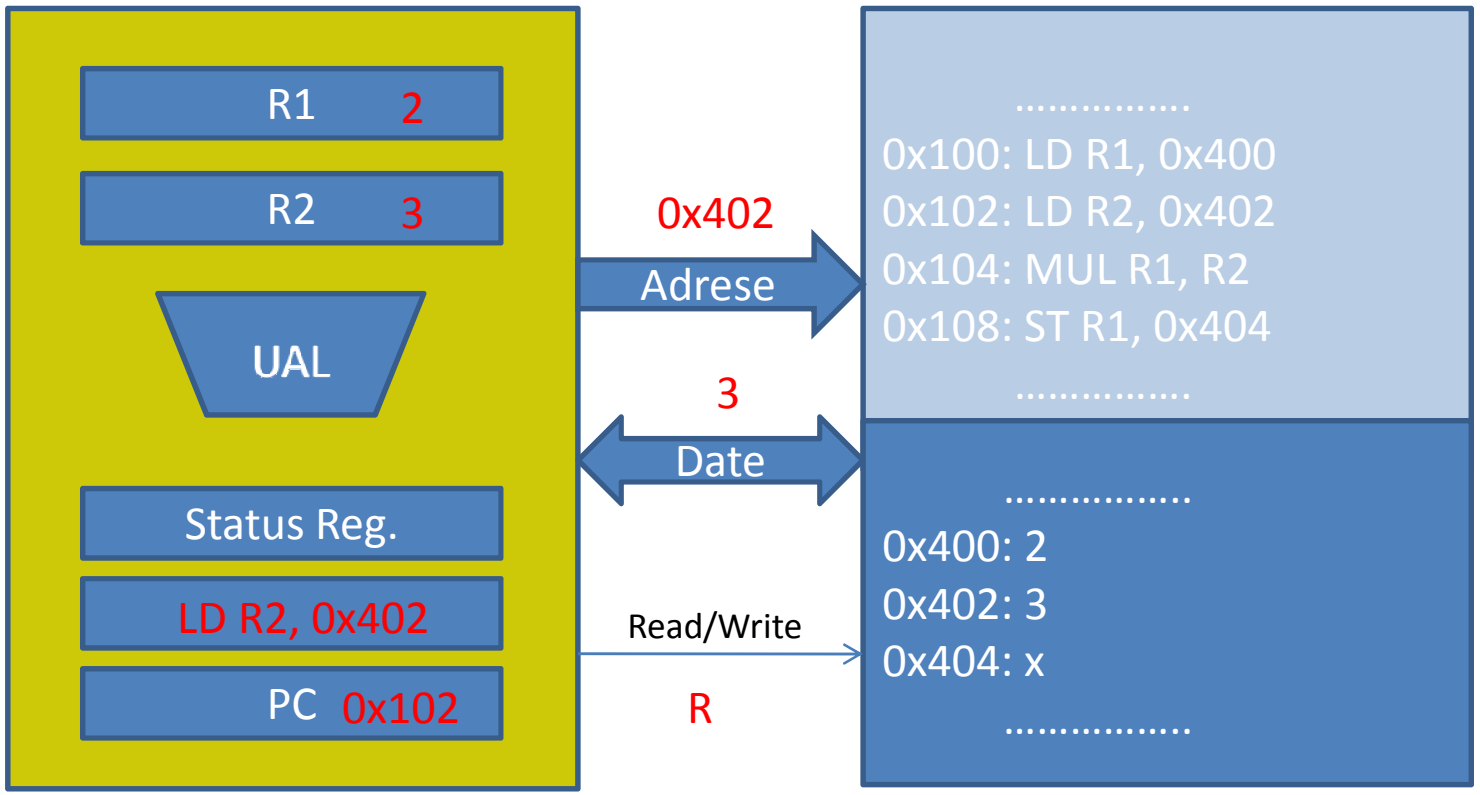
Arhitectura von Neumann

Fetch 0x102



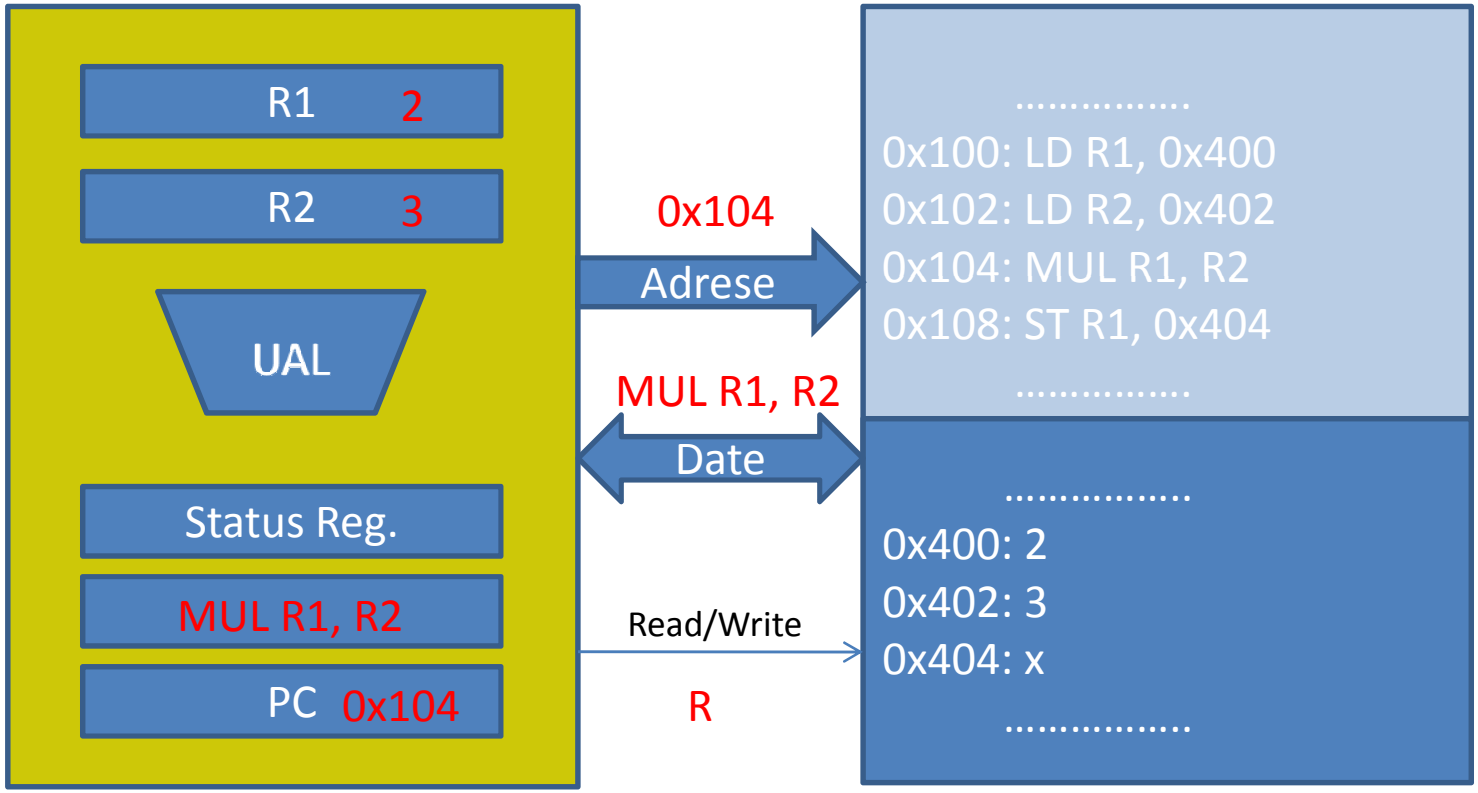
Arhitectura von Neumann

Execute 0x102



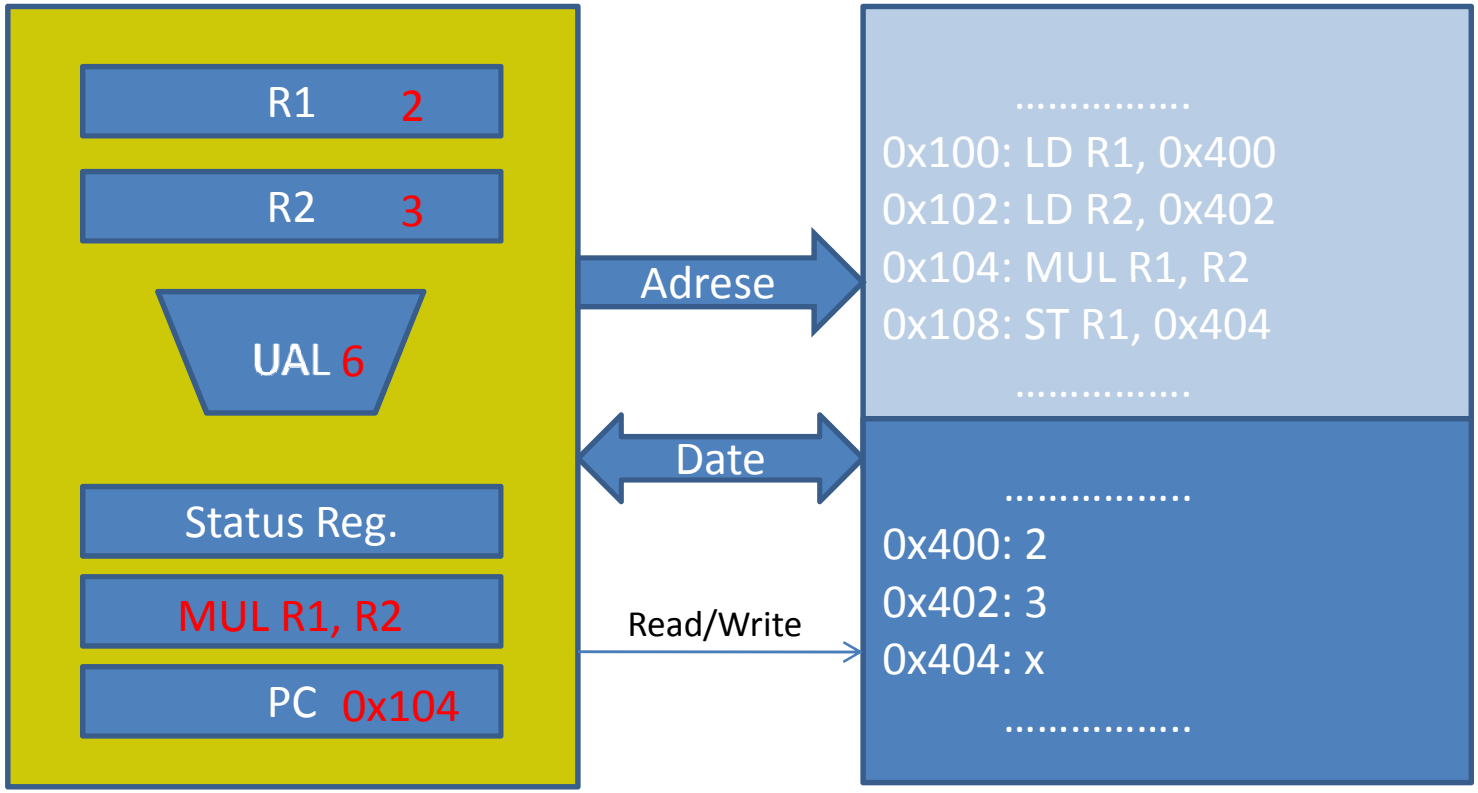
Arhitectura von Neumann

Fetch 0x104



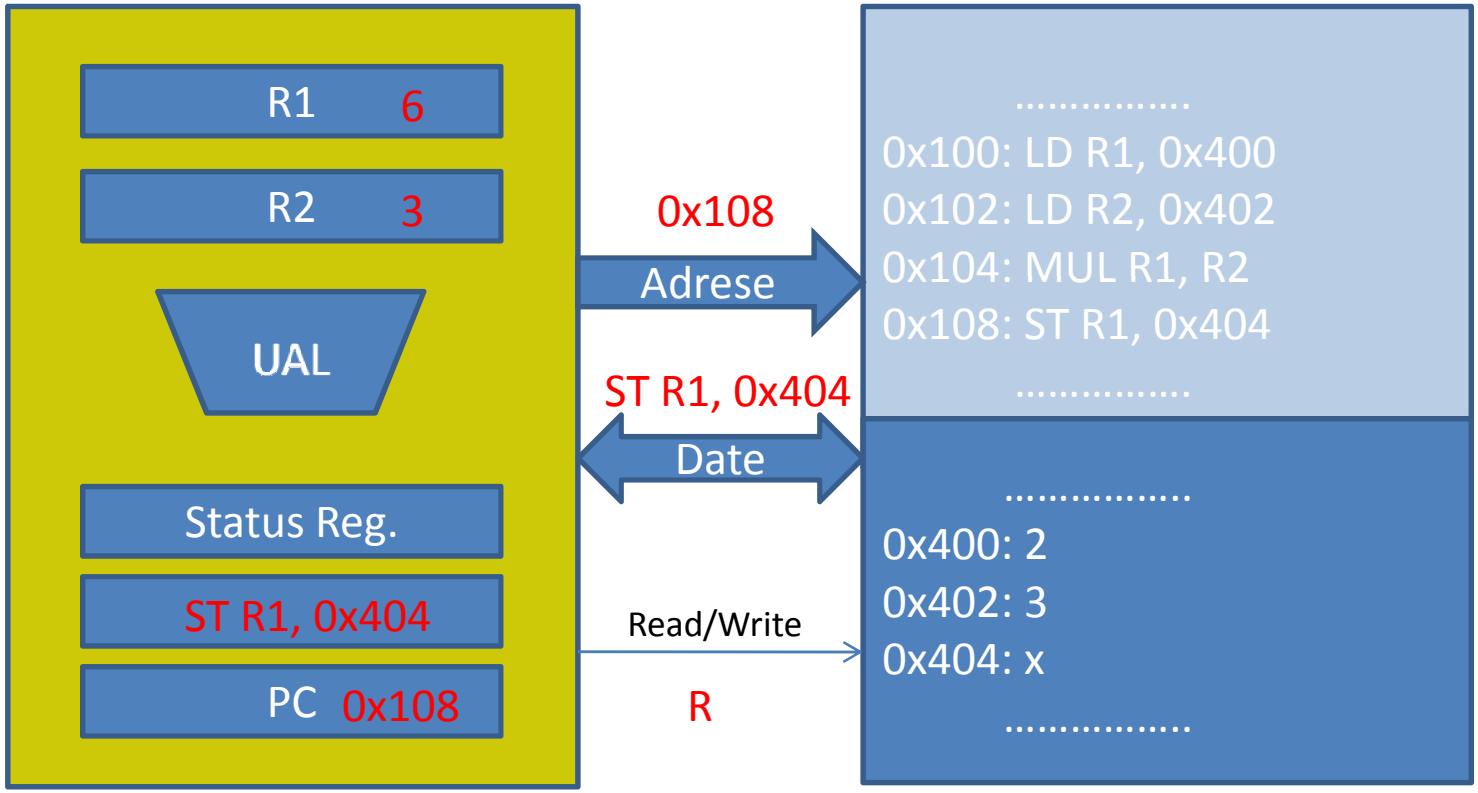
Arhitectura von Neumann

Execute 0x104



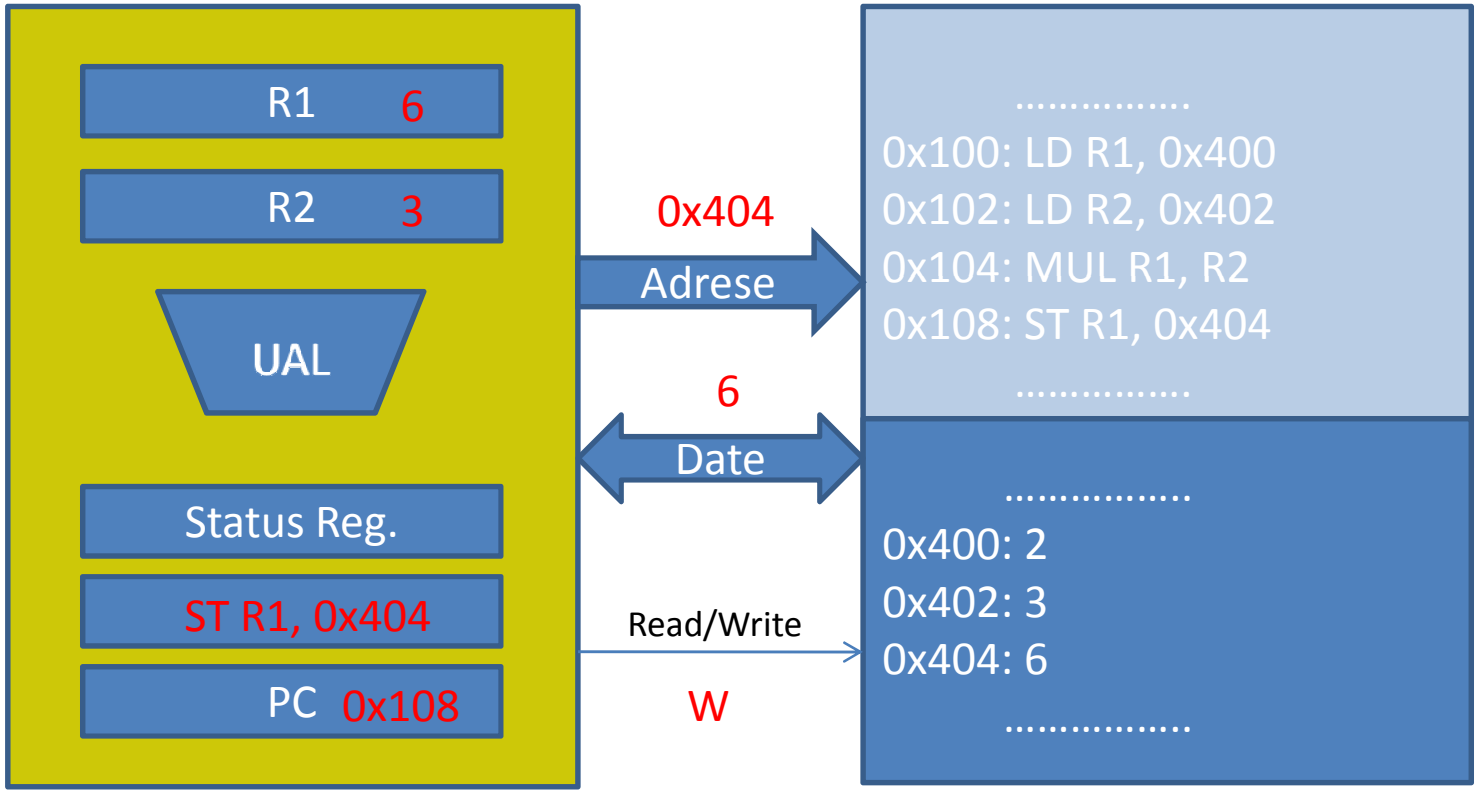
Arhitectura von Neumann

Fetch 0x108

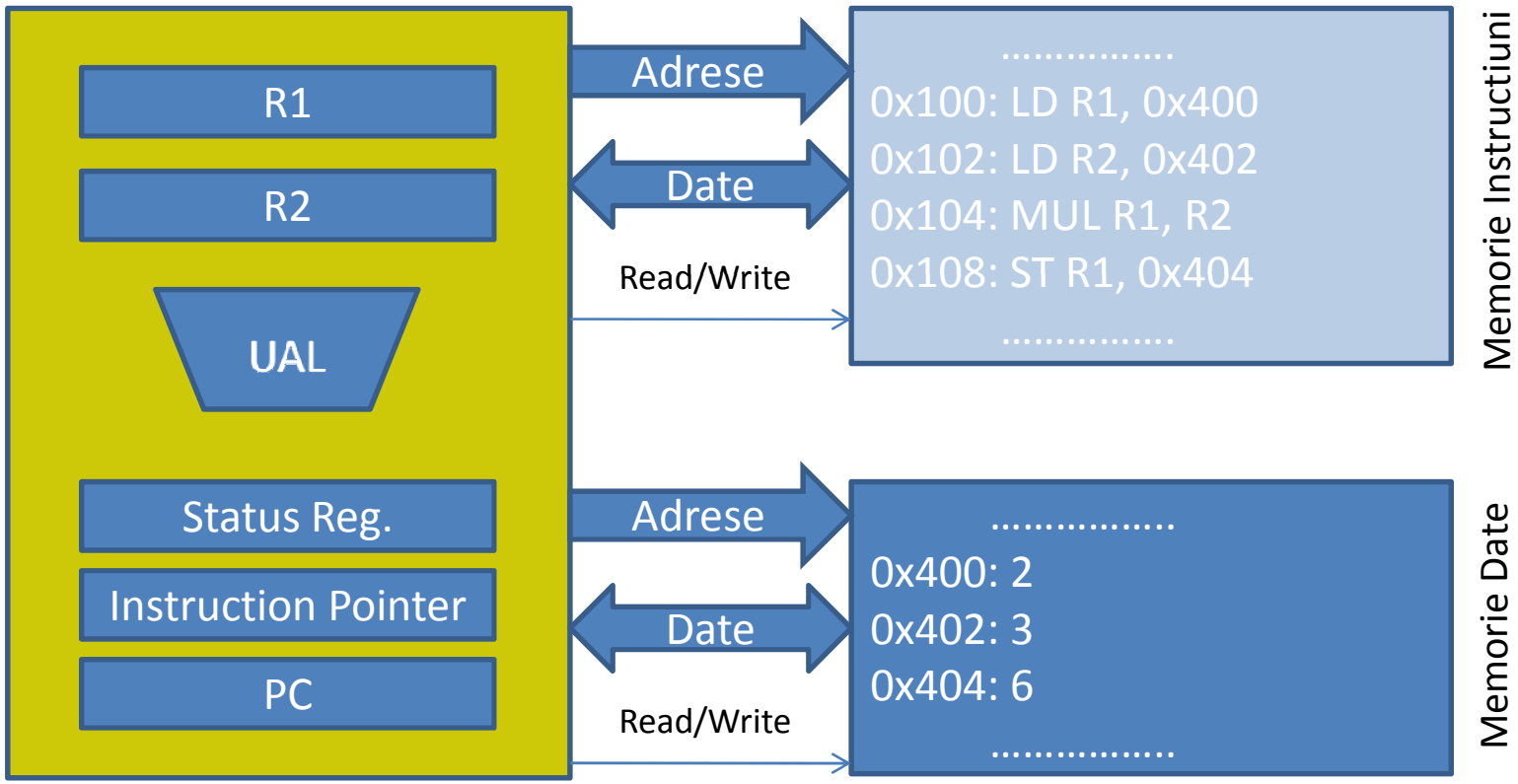


Arhitectura von Neumann

Execute 0x108



Arhitectura Harvard

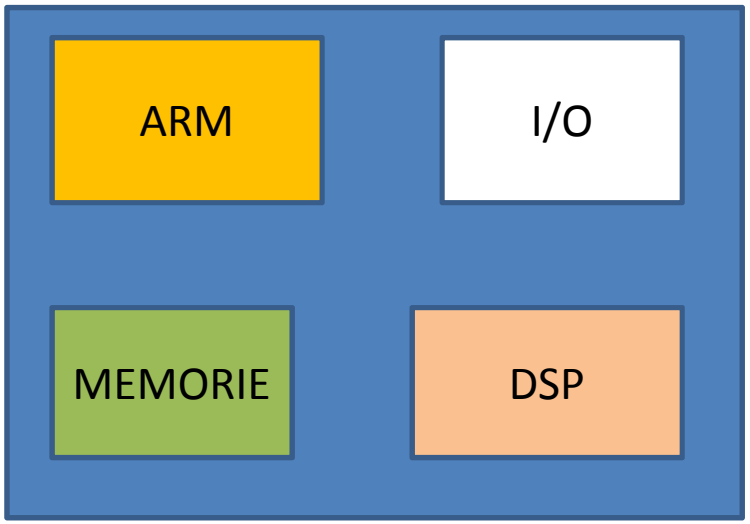


Arhitectura Harvard

- Memoria de date si memoria de program sunt separate
  - Pot sa fie tehnologii diferite (Flash, RAM, EPROM)
- Poate sa faca doi cicli de fetch simultan (unul din memoria de date si altul din memoria de program)
- Memoria de program nu poate fi modificata
  - Nu suporta cod care sa se auto-modifice
- Majoritatea DSP-urilor au arhitectura Harvard
  - Latime de banda mai mare
- Design mai complicat ca von Neumann
  - Hardware in plus pentru cele doua magistrale

Microprocesorul ARM

- Procesoarele ARM sunt o familie de arhitecturi RISC ce impartasesc aceleasi pricipii de design si un set comun de instructiuni.
- ARM nu produce procesoarele ci vinde licenta altor companii care integreaza procesorul in sisteme proprii.
- ARM este nucleul unui SoC (System on Chip)



ARM - Istoric

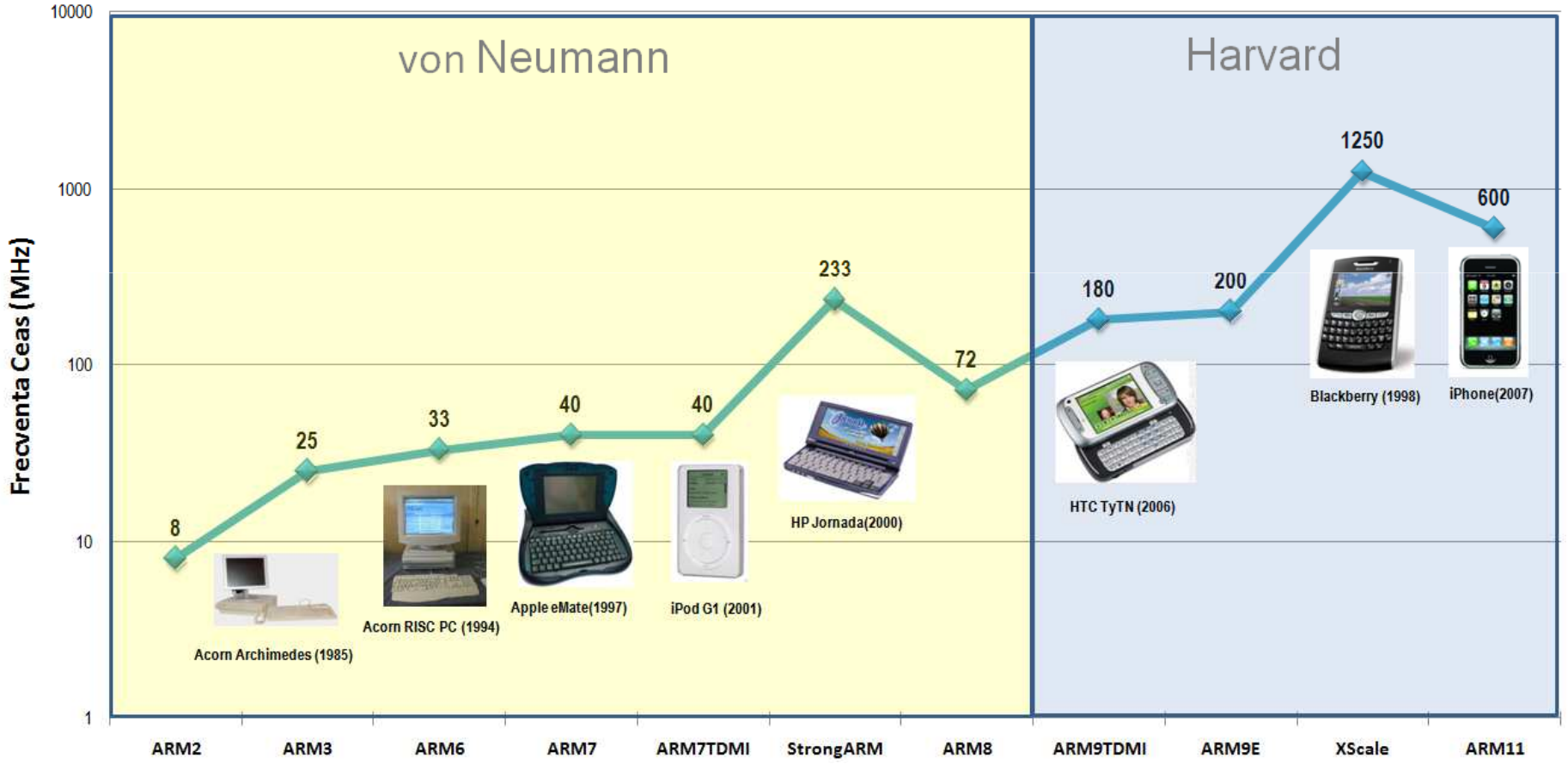
- 1978 – Acorn Computers Ltd. (Cambridge UK)
- 1985 – ARM1 (**A**corn **R**ISC **M**achine)
- 1986 – ARM2
  - Procesor pe 32 de biti
  - Fara memorie cache
  - Cel mai simplu procesor pe 32 de biti din lume (doar 30.000 de tranzistoare)
  - Performante superioare unui Intel 80286 contemporan
- 1991 – ARM6 (**A**dvanced **R**ISC **M**achine)
  - Consortiu Apple – Acorn
  - Primul PDA Apple (Newton)

ARM - Istoric

- 1995 – StrongARM
  - Produs DEC cu licenta ARM
  - Imbunatatiri substantiale de viteza

  si consum (1W la 233MHz)
- 1997 –Intel StrongARM
  - Achizitionat de la DEC
  - Menit sa inlocuiasca i860 si i960
- 2001 – ARM7TDMI
  - Cel mai popular procesor ARM
  - Incorporat in iPod, Game Boy Advance, Nintendo DS
- 2003 – ARM11
  - Nokia N95, iPhone
- 2006 – Marvell Xscale
  - Blackberry

ARM - Evolutie



Microprocesorul ARM

- Nucleul ARM
  - PDA
  - telefoane mobile
  - console de jocuri
  - orice alt instrument portabil.
- Costul redus de fabricatie si consumul mic de energie electrica au facut din ARM cel mai utilizat procesor din lume
  - peste 10 miliarde de procesoare vandute pana in 2008
  - 2011: 5 miliarde de procesoare anual
- Procesoarele variaza in functie de versiune si:
  - Memoria cache
  - Latimea magistralei
  - Frecventa de ceas

Microprocesorul ARM

- ARM este o arhitectura pe 32 de biti.

- Versiuni diferite pot utiliza arhitecturi diferite
  - ARM 7: von Neumann
  - ARM 9: Harvard

- Majoritatea chipurilor ARM implementeaza doua seturi de instructiuni
  - ARM Instruction Set (32 biti)
  - Thumb Instruction Set (16 biti)

- Jazelle core – procesorul executa bytecode Java
  - Majoritatea instructiunilor executate direct in hardware
  - Instructiunile complexe sunt executate in software
  - Compromis intre complexitate hardware / viteza de executie

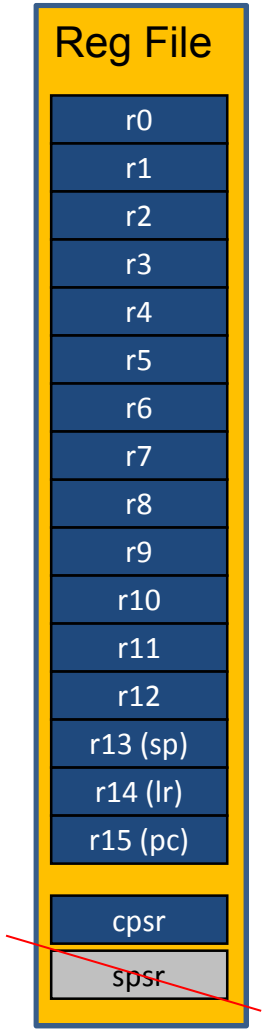
ARM – Moduri de operare

- ARM are sapte moduri de operare:
  - **User** : mod neprivilegiat; cele mai multe task-uri ruleaza aici
  - **FIQ** : declansat la producerea unei intreruperi de prioritate mare (fast interrupt)
  - **IRQ** : declansat la producerea unei intreruperi de prioritate normala (low interrupt)
  - **Supervisor** : declansare la RESET si la intreruperi software
  - **Abort** : la acces nepermis la memorie
  - **Undef** : pentru instructiuni nedefinite
  - **System** : mod privilegiat; foloseste aceleasi registre ca User Mode

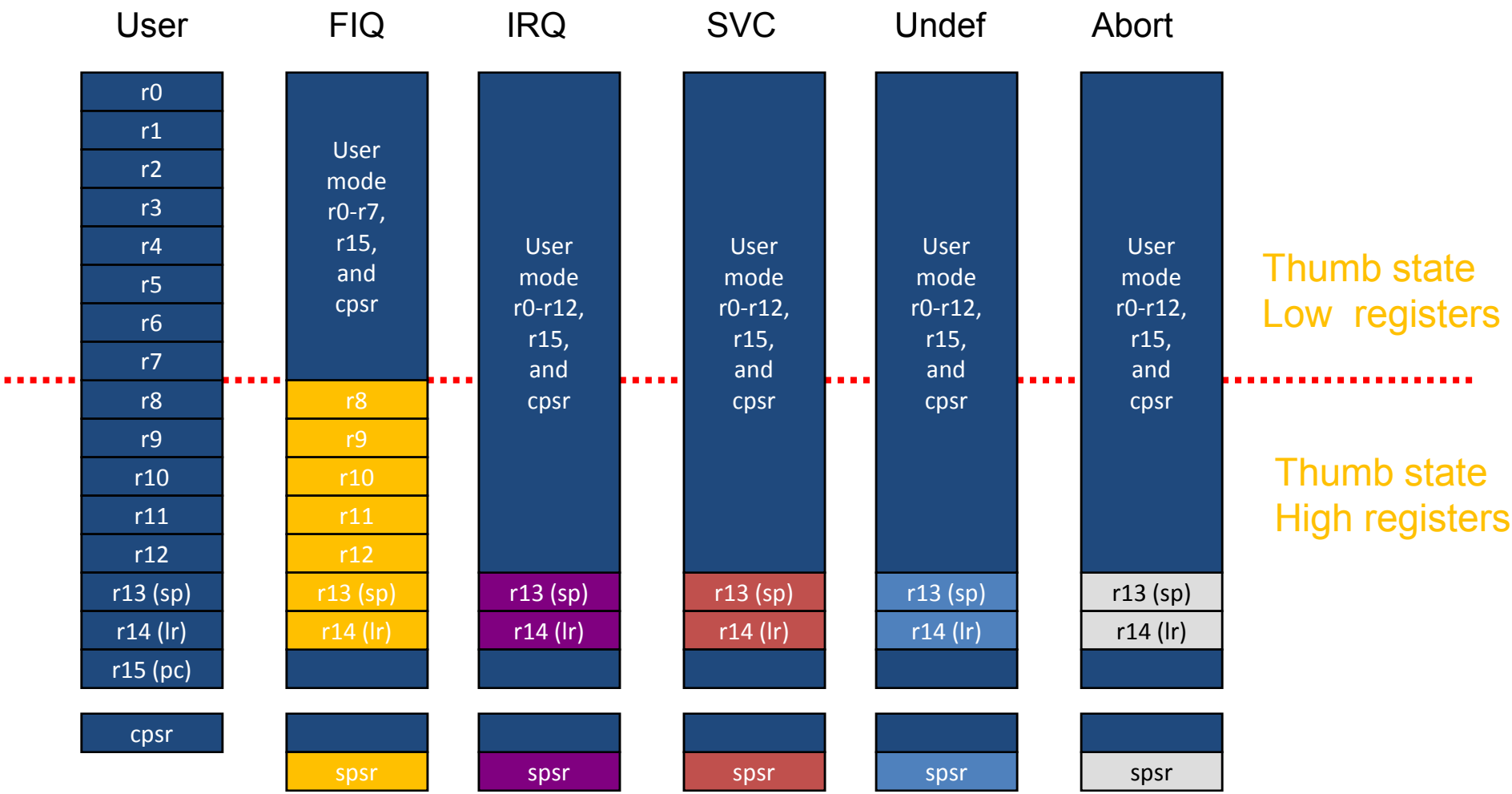
ARM – User Mode

Procesorul ARM are 17 registre active in User Mode

- 16 registre de date *(r0-r15)*
- 1 registru *processor status*
- *r13-r15 au si alte functii*
  - ✓ *r13 stack pointer (sp)*
  - ✓ *r14 link register (lr)*
  - ✓ *r15 program counter (pc)*
- *CPSR – Current Program Status Register*
- *SPSR - Saved Program Status Register*
  - ✓ *salveaza o copie a CPSR la producerea unei exceptii*
  - ✓ *nu este disponibil in User Mode*



ARM – Toate registrele



Nota: System mode foloseste acelasi set de registre ca si User mode

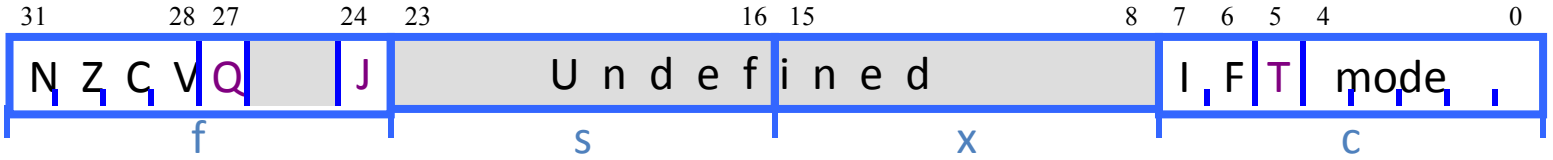
ARM - Registre

- ARM are 37 de registre de 32 de biti.
  - 1 program counter
  - 1 current program status register (CPSR)
  - 5 saved program status register (SPSR)
  - 30 registre general purpose

- Modul curent de lucru determina care banc de memorie este accesibil in acel moment. Indiferent de privilegii, fiecare mod poate accesa
  - registrele r0-r12
  - r13 (stack pointer, sp) si r14 (link register, lr)
  - registrul program counter, r15 (pc)
  - current program status register, cpsr

  Modurile privilegiate (mai putin System) pot sa acceseze si
  - un registru spsr (saved program status register)

ARM - CPSR



- Condition Flags
  - N = rezultat **N**egativ UAL
  - Z = rezultat **Z**ero UAL
  - C = **C**arry Flag
  - V = o**V**erflow flag
- Sticky Overflow Q – flag
  - Nu poate fi sters de alte operatii; decat printr-o instructiune dedicata.
- Bitul J
  - J = 1: Procesorul in starea Jazelle
- Interrupt Disable.
  - I = 1: IRQ Disable.
  - F = 1: FIQ Disable.
- Bitul T
  - T = 0: Procesor in starea ARM
  - T = 1: Procesor in starea Thumb
- Biti Mode
  - Specifica modul procesorului

ARM – Program Counter

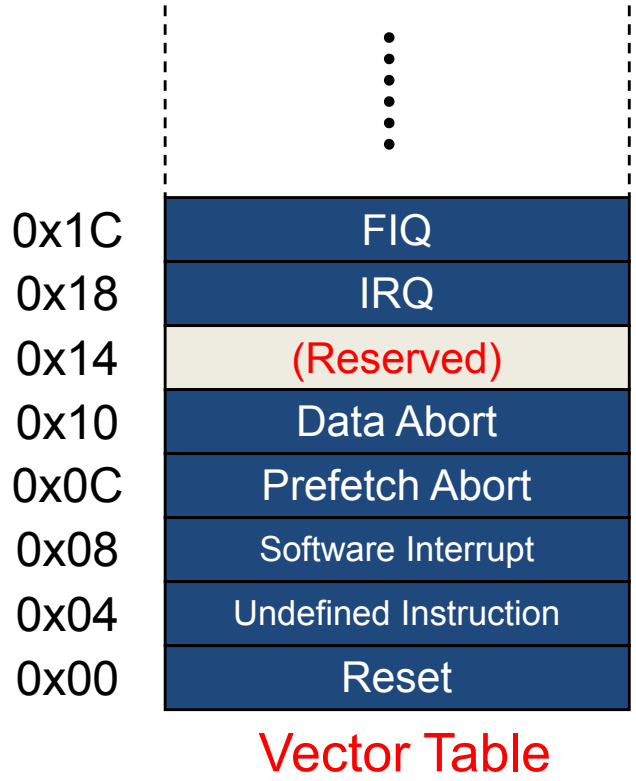
- Cand procesorul este in starea ARM:
  - Toate instructiunile au 32 de biti lungime
  - Instructiunile sutn aliniate la nivel de cuvant
  - Valoare PC este stocata in bitii [31:2] iar bitii [1:0] sunt nedefiniti.

- Cand procesorul este in starea Thumb:
  - Instructiunile au 16 biti lungime
  - Toate instructiunile sunt aliniate la nivel de jumatate de cuvant
  - Valoarea PC e stocata in bitii [31:1] cu bitul [0] nedefinit (instructiunile nu pot fi aliniate la nivel de octet).

- Cand procesorul este in starea Jazelle:
  - Toate instructiunile au 8 biti lungime
  - Procesorul citeste patru instructiuni simultan (citeste un cuvant de memorie)

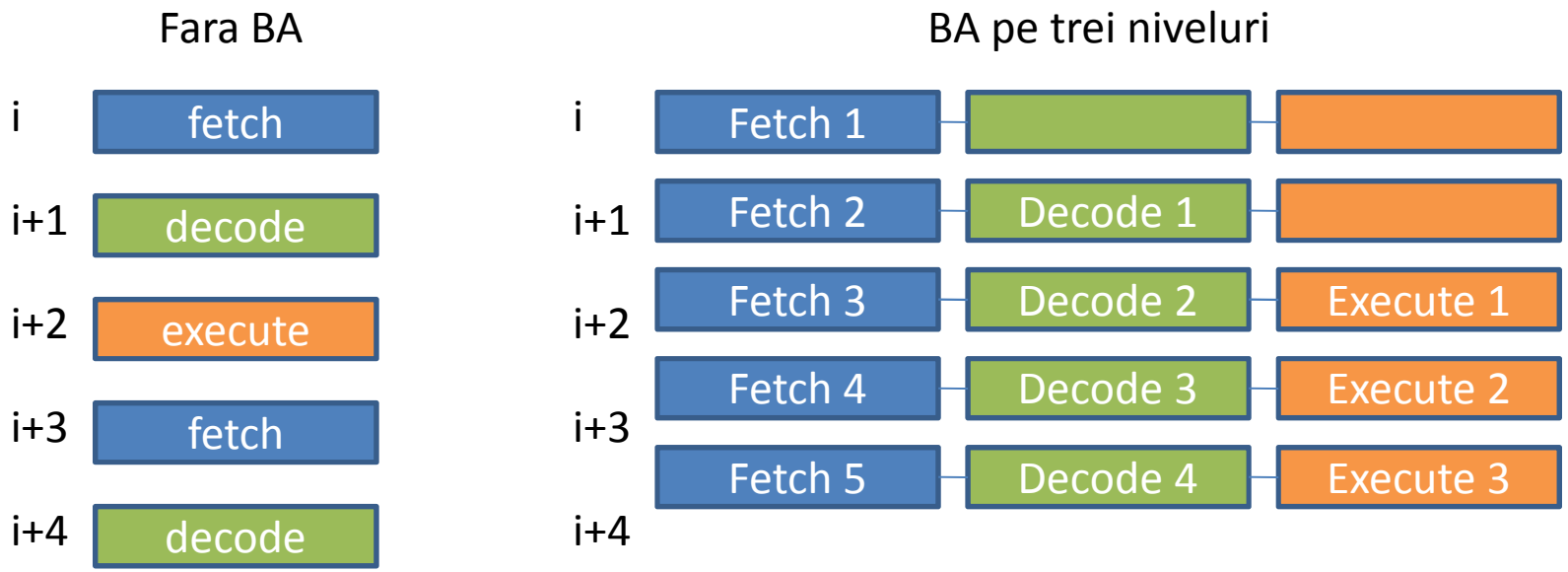
ARM – Tratarea Intreruperilor

- La aparitia unei exceptii:
  - Copiaza CPSR in registrul SPSR_<mod>
  - Seteaza bitii din CPSR
    - Intoarcere la starea ARM
    - Intrare in modul intrerupere
    - Opreste alte intreruperi (dupa caz)
  - Salveaza adresa de revenire in LR_<mod>
  - Seteaza PC la adresa vectorului curent
- La revenirea dintr-o intrerupere, un interrupt-handler:
  - Reface CPSR din SPSR_<mod>
  - Reface PC din LR_<mode>



ARM – Banda de Asamblare

- Banda de asamblare este folosita pentru a mari numarul de operatii executate per ciclu



ARM – Banda de Asamblare

- Banda de asamblare ARM7 are 3 niveluri

- Banda de asamblare ARM9 are 5 niveluri

ARM – Executie Conditionata

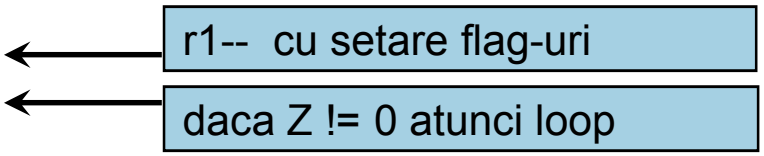
- Executia conditionala este realizata prin adaugarea de instructiuni care testeaza bitii de status dupa operatia curenta.
  - Reduce densitatea codului prin evitarea salturilor conditionale.

```
 CMP   r3,#0
  BEQ   skip
  ADD   r0,r1,r2
skip
```

```
CMP   r3,#0
 ADDNE r0,r1,r2
```

- In general, instructiunile de procesare a datelor nu afecteaza bitii de status. Acestia pot fi setati optional folosind sufixul “S”.

```
loop
  …
  SUBS r1,r1,#1
  BNE loop
```



ARM – Sufixe Conditionale

- Toate sufixele conditionale posibile:
  - AL este optiunea default; nu trebuie specificat

Sufix	Descriere	Flag-uri testate
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

ARM - Exemple

- Secventa de instructiuni conditionale

```
if (a==0) func(1);
    CMP     r0,#0
    MOVEQ   r0,#1
    BLEQ    func
```

- Seteaza flag conditie, apoi executa

```
if (a==0) x=0;
if (a>0)  x=1;
    CMP     r0,#0
    MOVEQ   r1,#0
    MOVGT   r1,#1
```

- Instructiuni de comparatie conditionale

```
if (a==4 || a==10) x=0;
    CMP     r0,#4
    CMPNE   r0,#10
    MOVEQ   r1,#0
```

ARM – Instructiuni de procesare date

- Sunt de 4 tipuri :
  - Aritmetice: `ADD` `ADC` `SUB` `SBC` `RSB` `RSC`
  - Logice: `AND` `ORR` `EOR` `BIC`
  - Comparatii: `CMP` `CMN` `TST` `TEQ`
  - Manipulare date: `MOV` `MVN`
- Functioneaza doar cu registre generale, nu si cu locatii de memorie.
- Sintaxa:

```
<Operatie>{<cond>}{S} Rd, Rn, Operand2
```

  - Comparatiile seteaza flag-urile – nu e specificat Rd
  - MOV, MVN nu specifica Rn

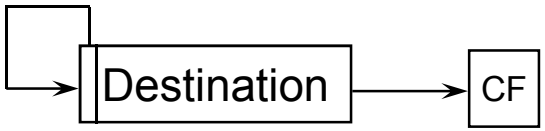
- Al doilea operand este trimis UAL prin Barrel Shifter.

ARM – Barrel Shifter

LSL : Logical Left Shift

Inmultire cu multipli de 2

ASR: Arithmetic Right Shift

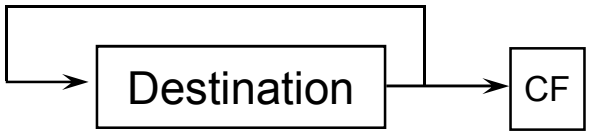


Impartire cu multiplu de 2
(pastreaza bitul de semn)

LSR : Logical Shift Right

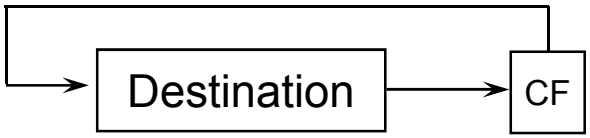
Impartire cu multiplu de 2

ROR: Rotate Right



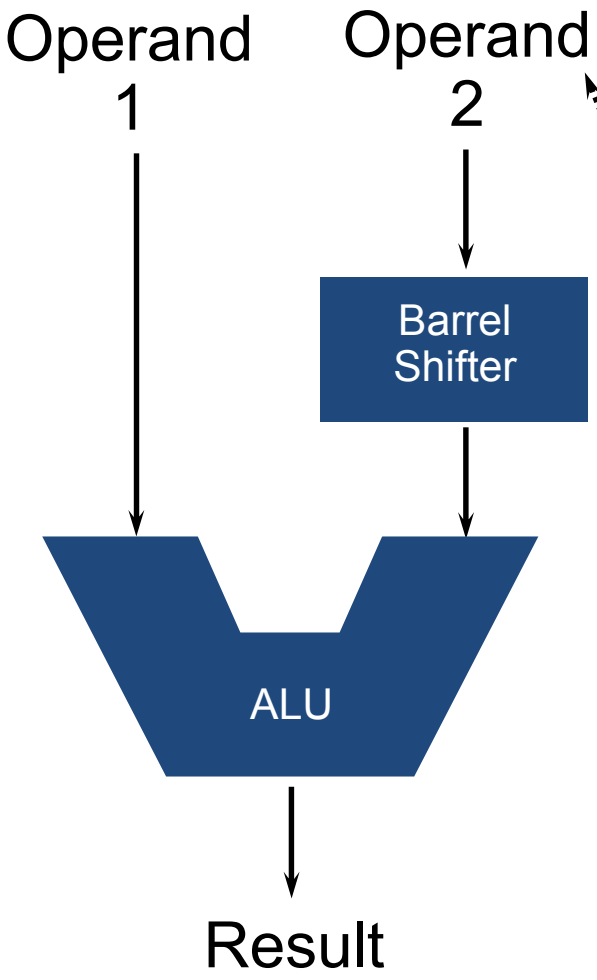
Rotatie aritmetica dreapta

RRX: Rotate Right Extended



Rotatie aritmetica dreapta extinsa
(8 biti + carry bit)

ARM – Barrel Shifter



Registru, optional cu operatie shiftare

- Numarul de rotatii:
  - intreg fara semn pe 5 biti
  - Ultimul octet al altui registru
- Folosit pentru inmultirea cu o constanta

Valoare imediata

- Constanta pe 8 biti
  - Rotita dreapta

Exemple:

```
ADD     r0, r1, r2
ADD     r0, r1, r2, LSL#7
ADD     r0, r1, r2, LSL r3
ADD     r0, r1, #0x4E
```

ARM – Lucrul cu Constante

- Asamblorul permite incarcarea de constante pe 32 de biti printr-o pseudo-instructiune speciala:
  - `LDR rd, =const`
- Pot fi doua rezultate:
  - O instructiune `MOV` care incarca valoarea in registru.

  sau
  - Genereaza un `LDR` cu o adresa indexata prin PC pentru a citi constanta dintr-o zona a codului dedicata constantelor (*literal pool*)
- De exemplu
  - `LDR r0,=0xFF` => `MOV r0,#0xFF`
  - `LDR r0,=0x55555555` =>

    ```
    LDR r0,[PC,#Imm12]
    …
    …
    DCD 0x55555555
    ```
- Este modalitatea recomandata de a incarca constante in cod

ARM - Inmultirea

- Sintaxa:
  - MUL{<cond>}{S} Rd, Rm, Rs — Rd = Rm * Rs
  - MLA{<cond>}{S} Rd,Rm,Rs,Rn — Rd = (Rm * Rs) + Rn
  - [U|S]MULL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := Rm*Rs
  - [U|S]MLAL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := (Rm*Rs)+RdHi,RdLo
- Cicli de instructiune
  - MUL
    - 2-5 cicli pentru ARM7TDMI
    - 1-3 cicli pentru StrongARM/XScale
    - 2 cicli pentru ARM9E/ARM102xE
  - +1 ciclu pentru ARM9TDMI (fata de ARM7TDMI)
  - +1 ciclu pentru inmultirea acumulata (MLA)
  - +1 ciclu pentru “long”

ARM – Load / Store

`LDR`	`STR`	Word
`LDRB`	`STRB`	Byte
`LDRH`	`STRH`	Halfword
`LDRSB`		Signed byte
`LDRSH`		Signed halfword

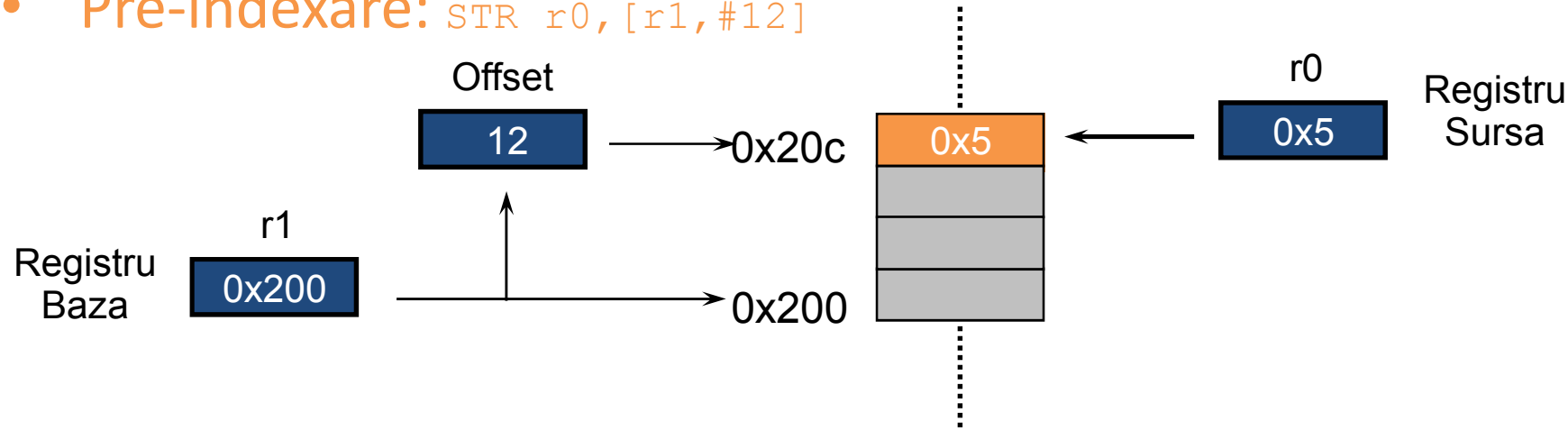
- Memoria trebuie sa suporte toate tipurile de acces.

- Sintaxa:
  - **`LDR`**{<cond>}{<size>} Rd, <address>
  - **`STR`**{<cond>}{<size>} Rd, <address>

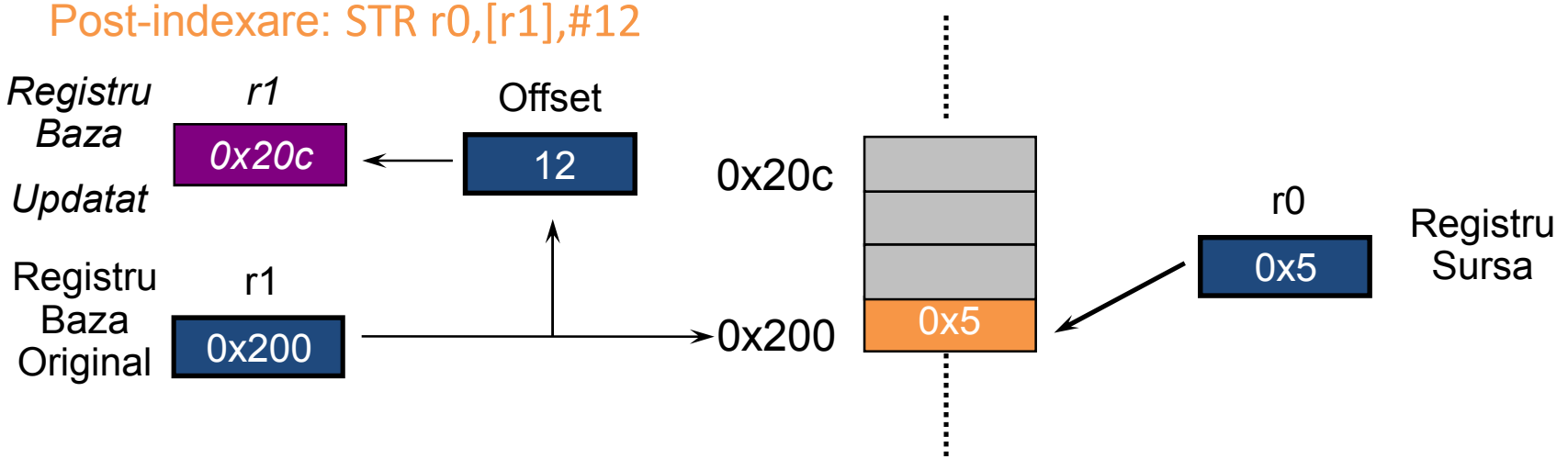
  e.g. **`LDREQB r0,[r1,#8]`**

ARM Load / Store

- Pre-indexare: `STR r0,[r1,#12]`

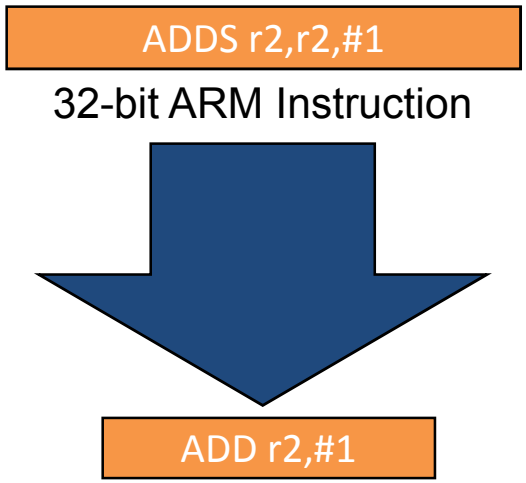


- Post-indexare: STR r0,[r1],#12



ARM - Thumb

- Thumb este un set de instructiuni pe 16 biti
  - Optimizat pentru cod C
  - Performanta marita chiar cu memorie ingusta
  - Subset de functii al setului de instructiuni ARM
- Nucleul procesorului are o stare speciala de executie – Thumb
  - Ciclare intre ARM si Thumb cu instructiunea **BX**

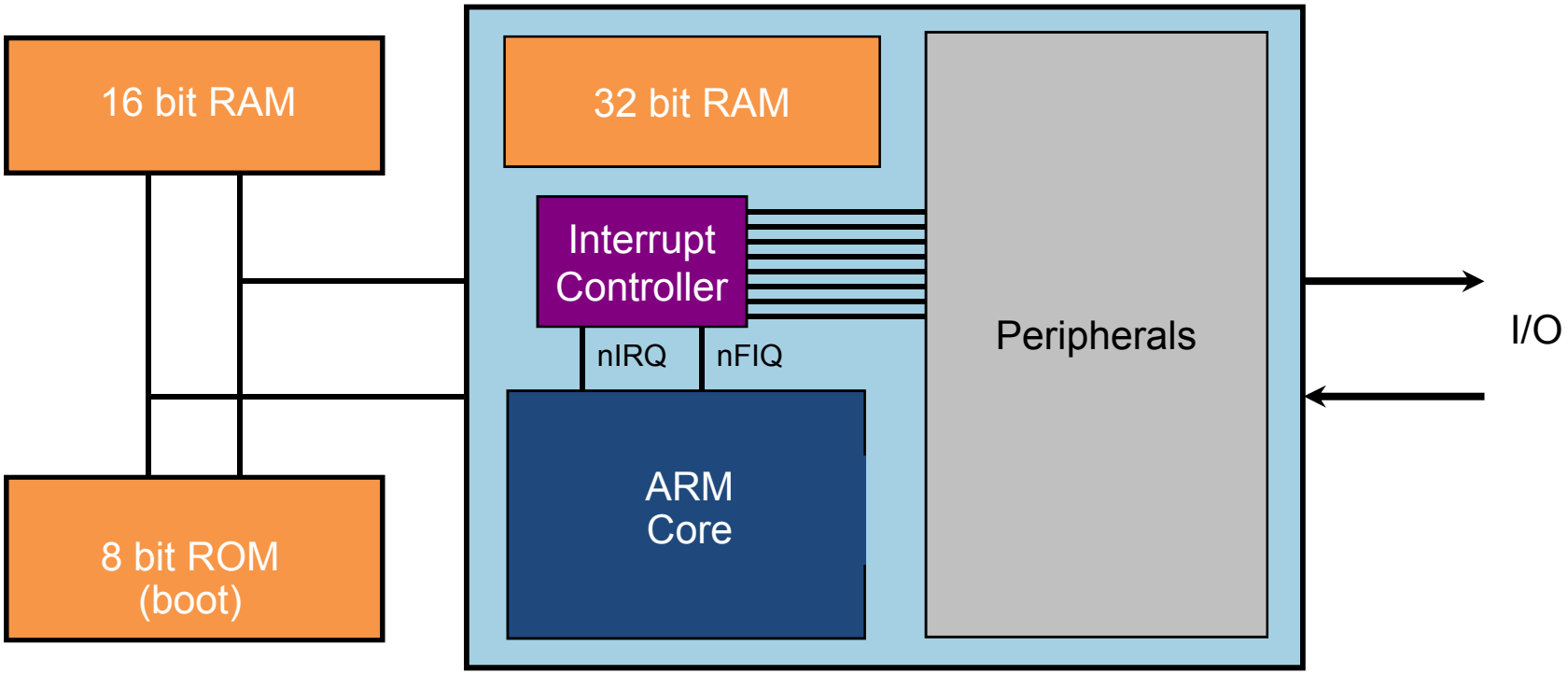


De ce Thumb?

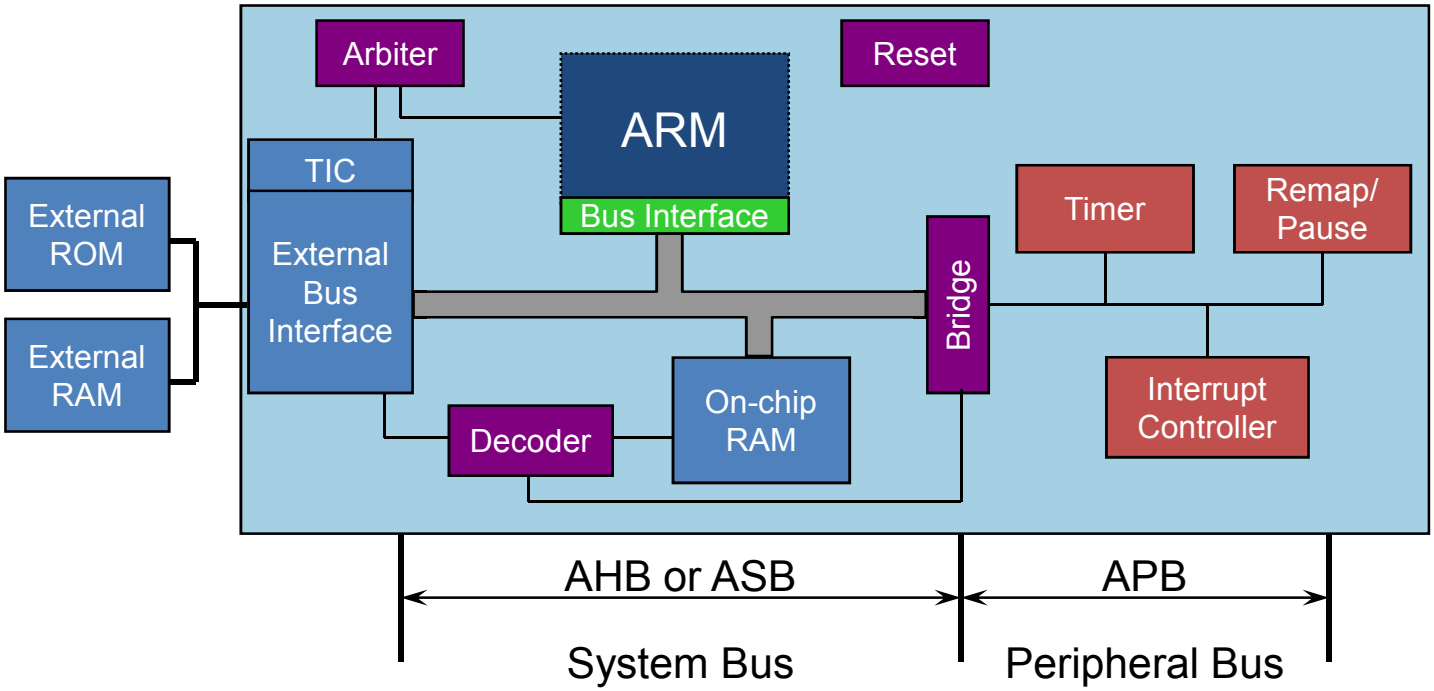
Pentru majoritatea instructiunilor generate de compilator:

- Executia conditionala nu e folosita
- Registrele sursa si destinatie sunt identice
- Doar ultimii 16 biti din registru sunt folositi
- Constantele au marime limitata

ARM – Exemplu de Arhitectura



ARM - AMBA



- AMBA (Advanced Microcontroller Bus Architecture)
  1. Standardul ARM de interconectare
  2. Refolosirea IP-core-ului in mai multe proiecte
  3. Permite facilitatea de upgrade si existenta mai multor familii de device-uri

Unde puteti intalni un ARM?

- Altera Corporation
- Analog Devices Inc
- Atmel
- Broadcom Corporation
- Conexant
- Dolby Labs
- Fraunhofer IIS
- Freescale Semiconductor
- IAR Systems
- Intel Corp
- Mentor Graphics Corporation
- Microsoft
- MSC Vertriebs GmbH
- National Semiconductor
- NEC Engineering
- Nokia
- NVIDIA
- ON Semiconductor
- Samsung Electronics
- Sanyo
- Sun Microsystems Inc
- STMicroelectronics
- Texas Instruments
- Toshiba
- Zilog

De ce procesoarele ARM sunt cele mai populare de pe piata?

- Low power
  - Solutiile embedded cu ARM ofera cel mai mic consum MIPS/Watt de pe piata la ora actuala.
  - De exemplu, familia STR7 cu ARM7TDMI: 10uA in Stand-by mode
- Cost redus al siliciului
  - Procesoarele ARM si alte produse IP folosesc eficient capabilitatile de procesare si memoria si sunt gandite pentru cerintele de pe piata dispozitivelor wireless.
- Nucleu de calcul performant
  - Arhitecturi de la 1MHz la 1GHz
  - O serie larga de solutii OS, Middleware si tool-uri de programare suportate
  - Foarte multe codecuri multimedia optimizate pentru ARM (ARM Connected Community)

Arhitectura AVR32

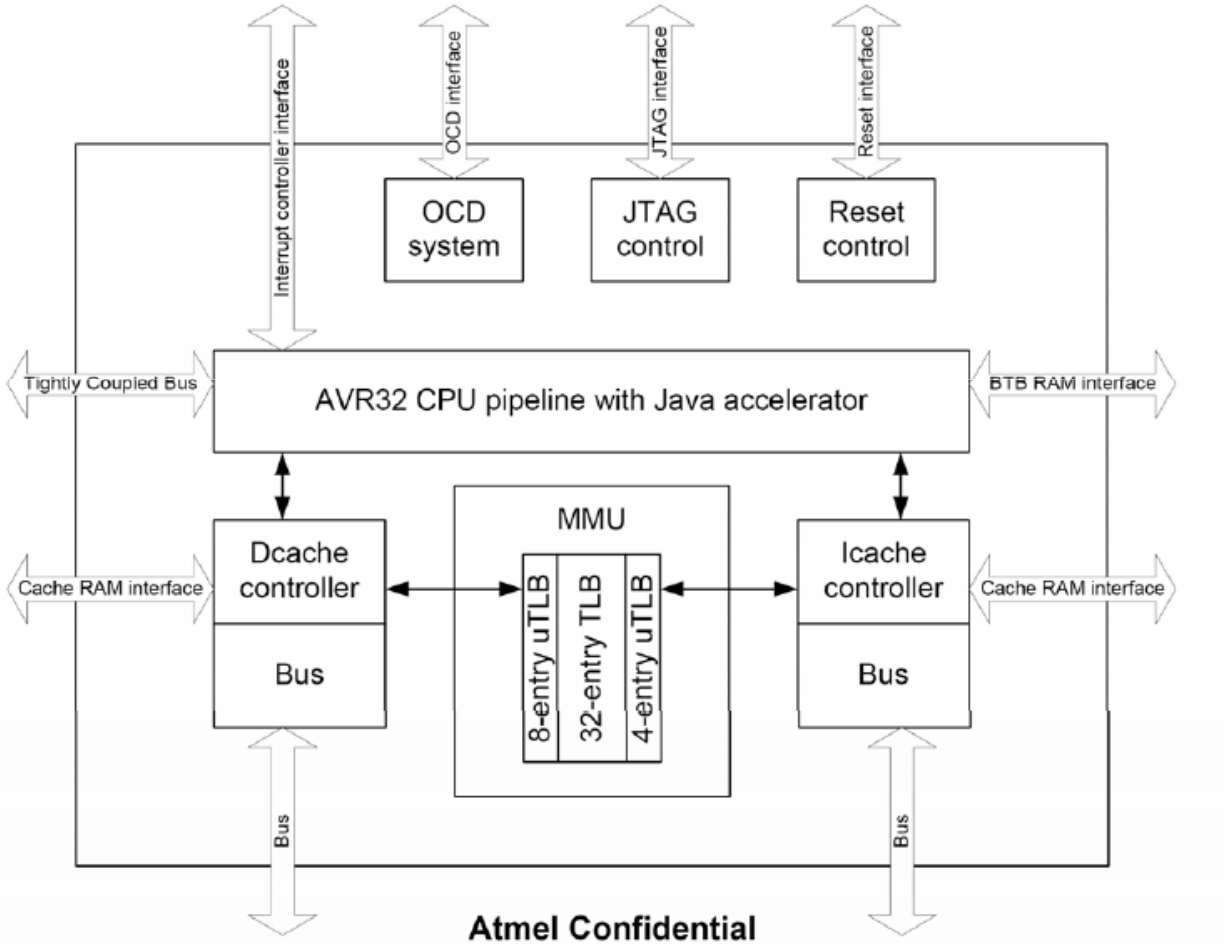
- NU este compatibil la nivel arhitectural cu AVR
- Formatul instructiunilor este flexibil pentru a obtine o densitate mare a codului.
  - Instructiuni simple de 16 biti si extinse de 32 de biti
- Instructiunile compacte si extinse pot fi amestecate liber in cod
- Pentru a reduce marimea codului la minim, unele instructiuni au moduri de adresare multiple
- Instructiunile utilizate frecvent, cum ar fi add, au un format compact cu doi operanzi precum si un format extins cu trei operanzi
- Formatul extins mareste performanta – permite efectuarea unei adunari si a unui transfer de date intr-un singur ciclu
- Instructiunile Load/Store au formate diferite pentru a reduce marimea codului si a creste viteza de executie
  - –Load/store to an address specified by a pointer register
  - –Load/store to an address specified by a pointer register with postincrement, predecrement, and displacement
  - –Load/store to an address specified by a small immediate (direct addressing within a small page)
  - –Load/store to an address specified by a pointer register and an index register

Arhitectura AVR32

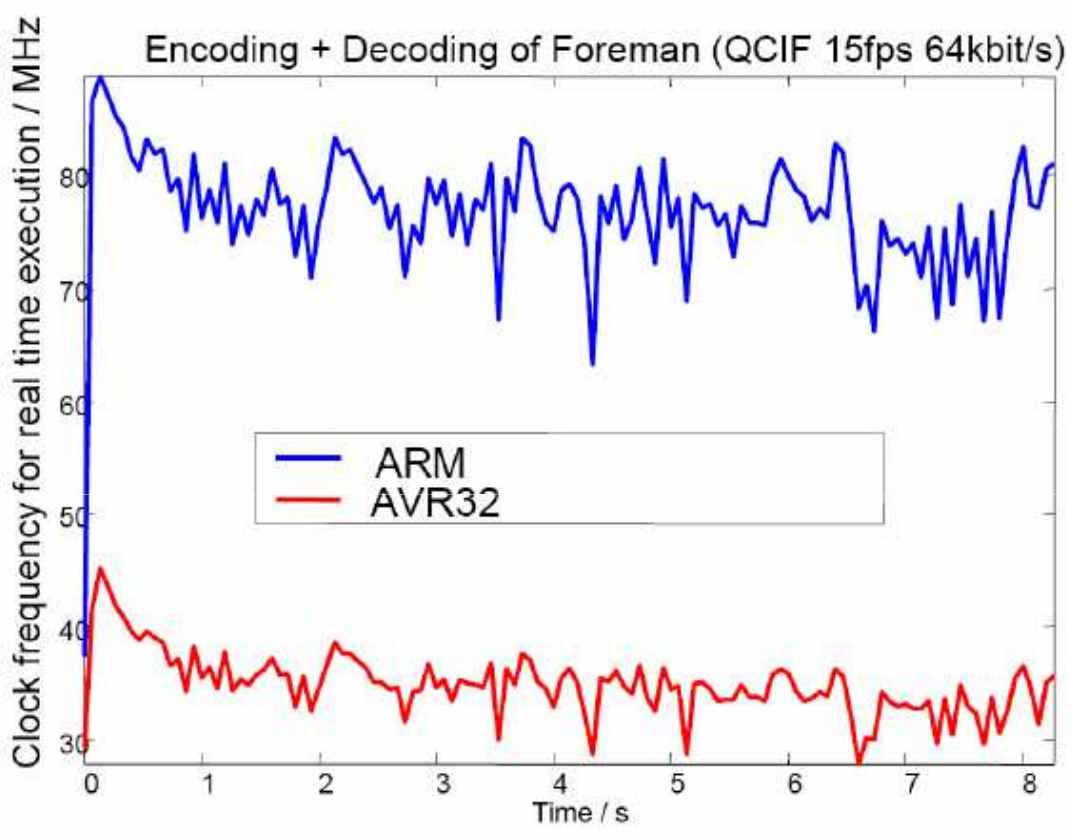
- Event handling
  - –The different event sources have different priority levels, ensuring a well-defined behavior when multiple events are received simultaneously.
  - –Pending events of a higher priority class may preempt handling of ongoing events of a lower priority class
  - –Each priority class has dedicated registers to keep the return address and status register thereby removing the need to perform time-consuming memory operations to save information
  - –4 level external interrupts
- Microarchitectures
  - –AVR32A: cost-sensitive, lower-end application – similar cu microcontrollerele pe 8biti.
  - –Does not provide dedicated hardware registers for shadowing of register file registers in interrupt context and hardware registers for the return registers and return status register

    All information are stored on the system stack
  - –AVR32B: be targeted at applications where interrupt latency is important.
  - –Implements dedicated registers to hold the status register and return address for interrupts, exceptions and supervisor calls this information does not need to be written to the stack, and latency is therefore reduced
  - –The INT0 to INT3 contexts may have dedicated versions of the registers in the register file

AVR32 vs. ARM


Interrupt controller interface
OCD interface
JTAG interface
Reset interface
OCD system
JTAG control
Reset control
Tightly Coupled Bus
AVR32 CPU pipeline with Java accelerator
BTB RAM interface
MMU
Dcache controller
Bus
8-entry uTLB
32-entry TLB
4-entry uTLB
Icache controller
Bus
Cache RAM interface
Cache RAM interface
Bus
Bus
Atmel Confidential

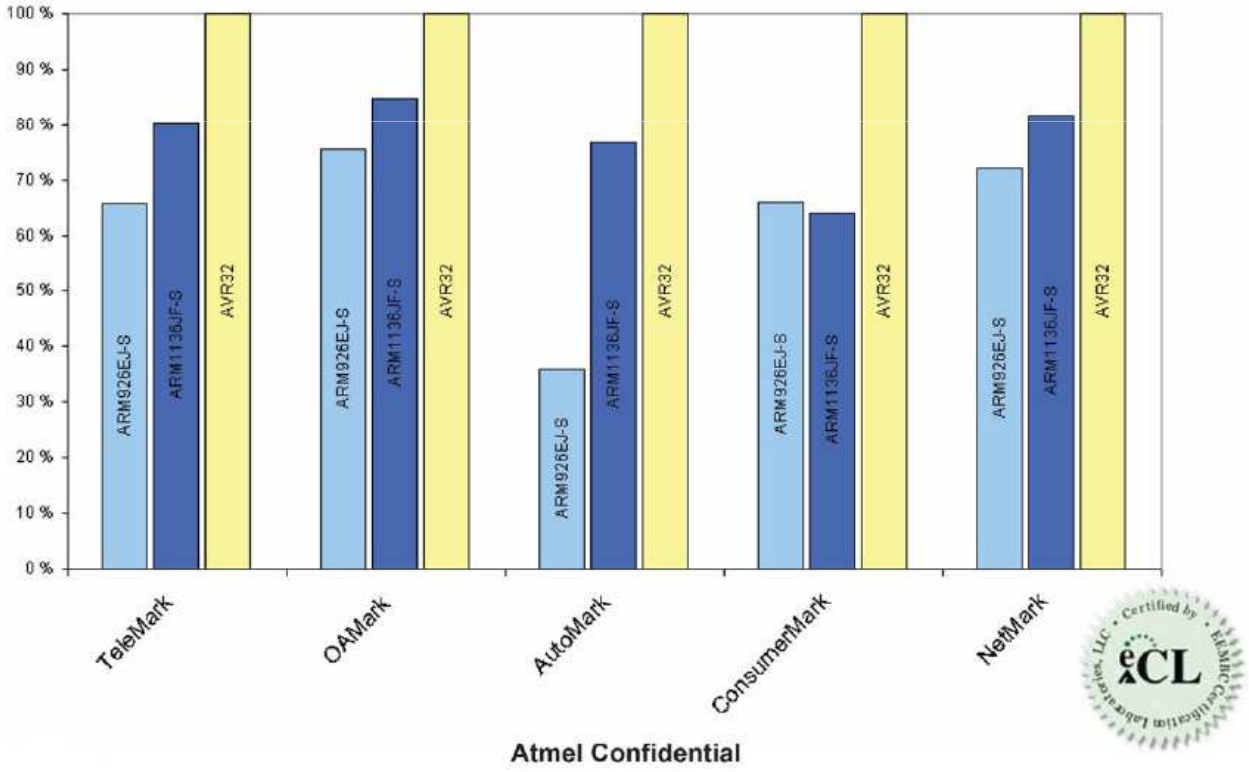
AVR32 Benchmark



- QVGA@30fps MPEG4 Decode: 75MHz CPU frequency
- MP3 Audio: 15MHz CPU frequency
- Depaseste ARM9 de 3 ori–video decode

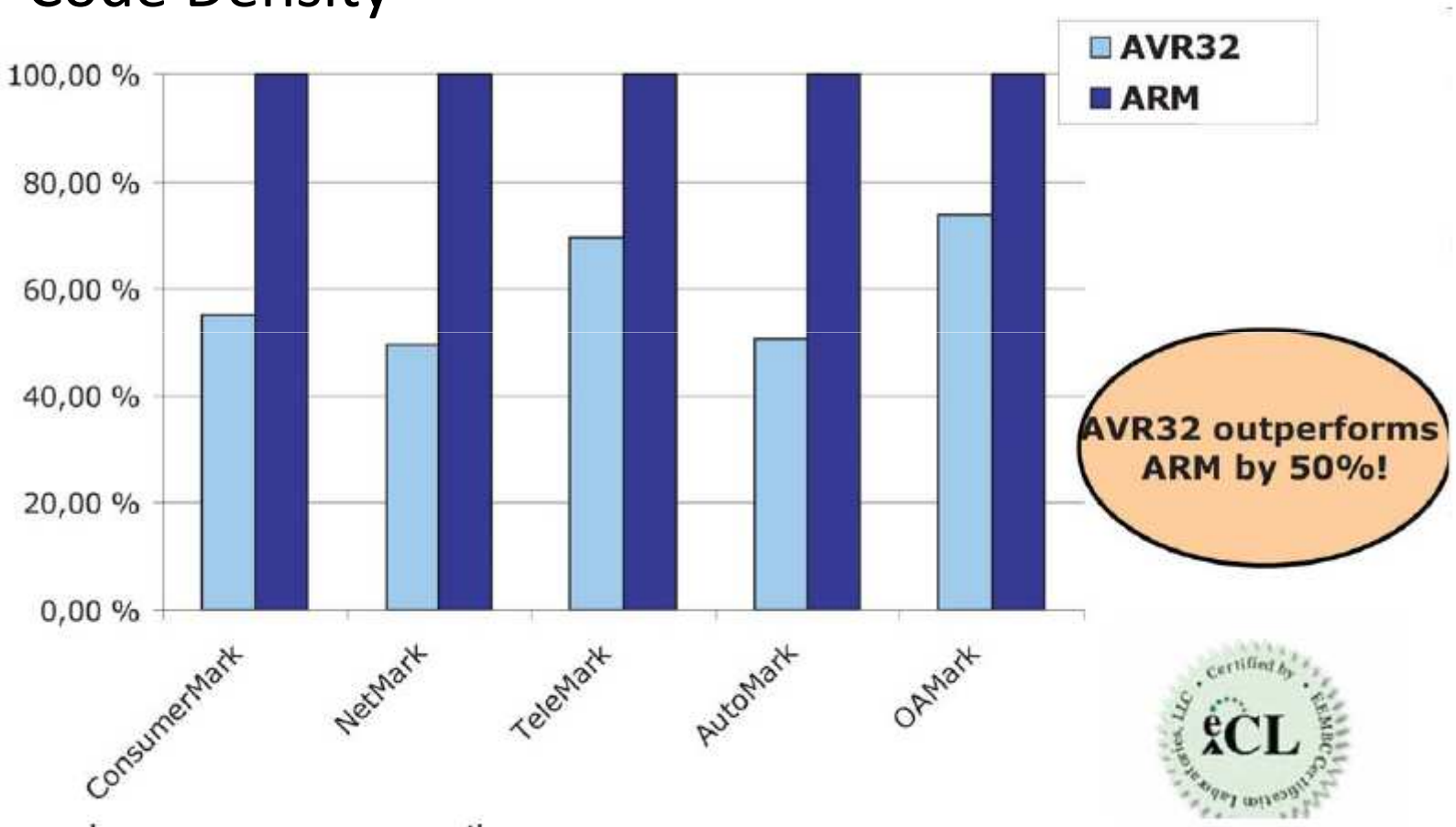
EEMBC –Generic Benchmarks

- Embedded Microcontroller Benchmarks Consortium
- Benchmark pentru arhitecturi, nu pentru un device anume



EEMBC –Generic Benchmarks(cont.)

- Code Density



  - Lower power consumption
  - Lower RAM requirement