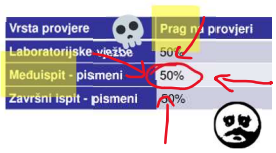
OBLIKOVANJE PROGRAMSKE POTPORE ZA UGRADBENE RAČUNALNE SUSTAVE



Međuispit

Predavanja – prezentacije

1. Arhitektura i projektiranje složenih ugradbenih računalnih sustava

Ugradbeni računalni sustav

- računalni sustav opće namjene
  - širok raspon mogućih aplikacija
  - korisnik može proširivati funkcionalnost
  - poboljšanje performansi – proširenjem sklopovlja
- ugradbeni računalni sustav
  - ciljana aplikacija – specifična funkcija
  - korisnik ne reprogramira – nije opće namjene
  - poznati zahtjevi – bez nepotrebnog sklopovlja

- zahtjevi ugradbenih računalnih sustava:
  - **funkcionalnost** – obavljanje složenog algoritma i ostvarivanje korisničkog sučelja
  - **rad u stvarnom vremenu** – deterministički garantiran odziv u propisanim vremenskim okvirima
    - pravovremenost reakcije je važnija od same brzine odziva
  - **učinkovitost** – proizvodna cijena, niska potrošnja, veličina koda, brzina izvođenja, masa, dimenzije…
  - **pouzdanost**

    hardware + software co-design – odabrati optimalnu sklopovsku i programsku tehnologiju za projekt

- oblikovanje programske potpore:
  - **„bare-metal"** – bez operacijskog sustava (aplikacijski kod + device drivers)
  - **real-time operating system** (RTOS)
  - **general-purpose operating system** (GPOS)

  ⟹ zahtjevaju značajniju sklopovsku podršku (procesorska snaga, memorijski resursi, i sl.)

Mikroprocesor i mikrokontroler

- **mikroprocesor** (CPU)
  - CISC/RISC arhitekture (complex/reduced); 8,16,32,64-bitni
  - ARM, AVR, Blackfin, MIPS, SPARC, PowerPC, x86 (CISC)…
  - ARM procesori – porodice, arhitekture, jezgre: ARM7TDMI, ARM720T, Cortex-M0, Cortex-M3…

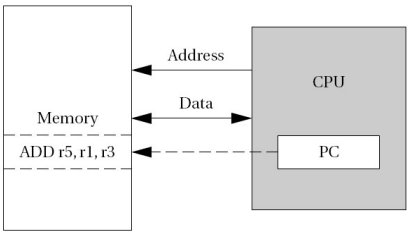
- **mikrokontroler** (MCU, µC)
  - računalni sustav koji unutar istog sklopa sadrži CPU, memoriju i ulazno-izlazne jedinice
  - STM32F, ATtiny, ATmega, MCS-51 (8051 family)…
  - 8-bit → 32-bit: veličina koda↘, performanse↗, energetska efikasnost↗, veličina↗, cijena↗

- **digital signal processor** (DSP)
  - specijalizirani mikroprocesor optimiran za algoritme digitalne obrade signala
  - sklopovsko modulo adresiranje, Harvardska arhitektura (streaming), SIMD, brz multiply-accumulate…
  - ADSP-21xx, Blackfin, FPGA rješenja

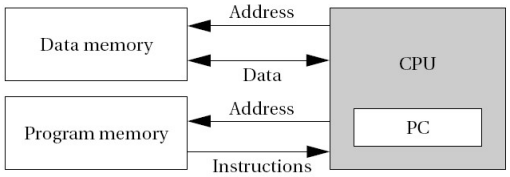
- **soft procesor**
  - implementiran u programabilnoj logici - FPGA
  - fleksibilnost (future-proofing), ali skupo i neefikasno (velika potrošnja)
  - PicoBlaze, MicroBlaze…

- ARM Cortex arhitektura:
  - **load/store** (`R0-R15` + xPSR), **Harvard** (`Icode` i `Dcode` bus), memory-mapped I/O
  - unificirana memorijska mapa – Code 0x00000000, SRAM 0x20000000, Peripheral 0x40000000, ...

von Neumann arhitektura

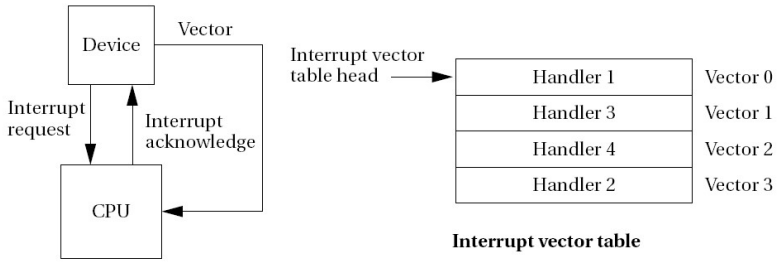


Harvard arhitektura



- pristup **ulazno-izlaznim jedinicama**
  - **memory-mapped I/O** – u zajedničkom memorijskom prostoru
  - **port-mapped I/O** – u posebnom adresnom prostoru, zasebne instrukcije za pristup
  - programski pristup: hardware abstraction layer (HAL) rutine + device drivers
    - **prozivanje** (busy-wait, polling) – neefikasnost iskorištenja procesora
    - ili **prekid**

- prekidi:
  - okolina od procesora zahtjeva pozornost („paralelizam“ izvođenja operacija)
  - interrupt vector table – sadrži adrese ISR (interrupt service routine)
  - maskiranje – privremeno onemogućavanje određenih prekida
  - prioriteti – neki prekidi su važniji
  - non-maskable interrupt (NMI)

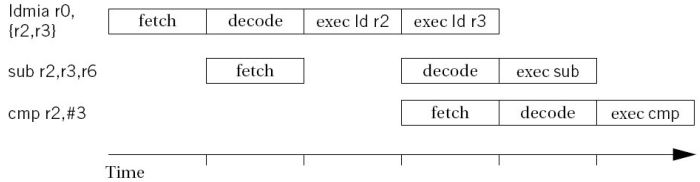


- načini rada procesora:
  - user mode – procesor izvodi aplikaciju
  - supervisor mode – privilegirane instrukcije (jezgra operacijskog sustava, zamjena konteksta dretve)

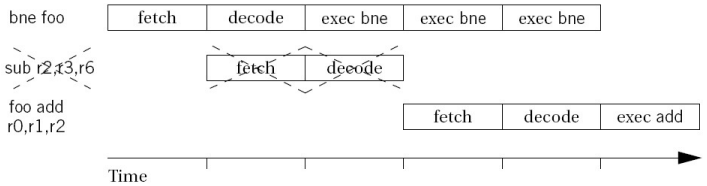
- iznimke:
  - sinkrone – uzrokovane izvođenjem programa (npr. greške u izvođenju instrukcije, zahtjev za privilegijom)
  - asinkrone – vanjski događaji (to su zapravo prekidi!)
  - ISR/ESR vector table – postavlja se prilikom inicijalizacije sustava (startup rutina)

- pipelining:
  - fetch → decode → execute – paralelno izvođenje više instrukcija unaprijed – poboljšanje performanse
  - data stall – čekanje podatka iz rezultata prethodne instrukcije
  - control stall – pogrešno predviđena sljedeća instrukcija

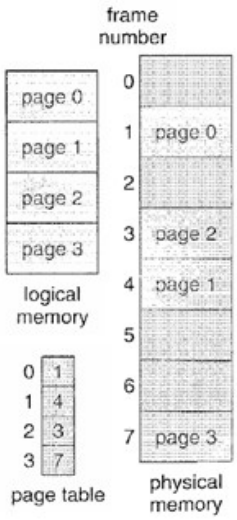
pipeline data stall



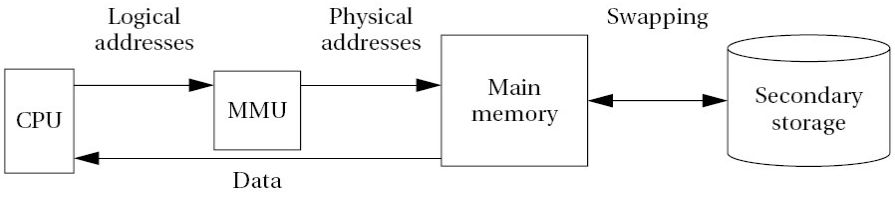
pipeline control stall



Sabirnica, memorije i periferija



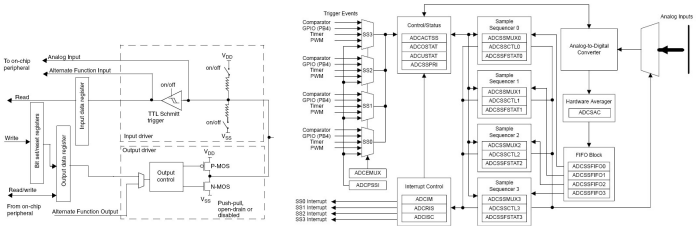
- **sabirnica** (bus)
  - komunikacijski put između CPU, memorije i ulazno-izlaznih jedinica
  - Memory Protection Unit – zaštita od upotrebe nevažećih ili zaštićenih adresa
  - Direct Memory Access (DMA) – transfer na sabirnici bez sudjelovanja procesora
  - Memory Management Unit (MMU) – logičko adresiranje i translacija adresa
    - straničenje – program vidi logičke adrese dok se OS brine o fizičkom rasporedu
    - omogućuje višezadaćnost



- **volatile memorija** (RAM)
  - SRAM (static) – skuplji, jednostavniji, manji kapacitet, niska potrošnja
  - DRAM (dynamic) – jeftiniji, veći kapacitet, visoka potrošnja (konstantno osvježavanje)

- **non-volatile memorija**
  - EEPROM – brisanje i prepisivanje na razini bajta, obično vanjska (SPI ili I^2C)
  - Flash – brisanje i prepisivanje u blokovima, vrste NOR ili NAND, veći kapacitet i niža cijena od EEPROM-a
    - varijante: MMC, CompactFLASH, SDCard (SPI)
  - NVRAM – non-volatile RAM (battery-backed)

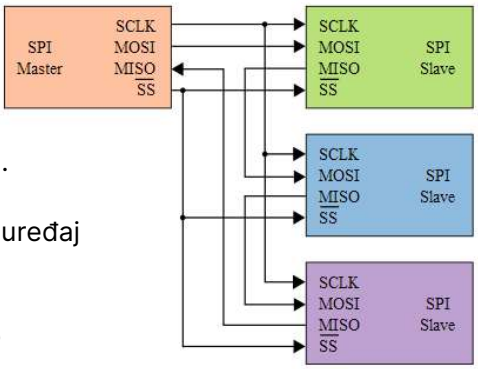
- ulazno-izlazne jedinice:
  - **general-purpose IO** (**GPIO**)
  - **A/D i D/A pretvornici** (**ADC** i **DAC**)



- izvedba vremenskih sklopova:
  - timer (generiranje vremenskih intervala, okidanje ADC-a, real-time clock…) + counter + compare
  - mogućnost generiranja PWM signala
  - Watchdog – automatski reset sustava u slučaju zastoja

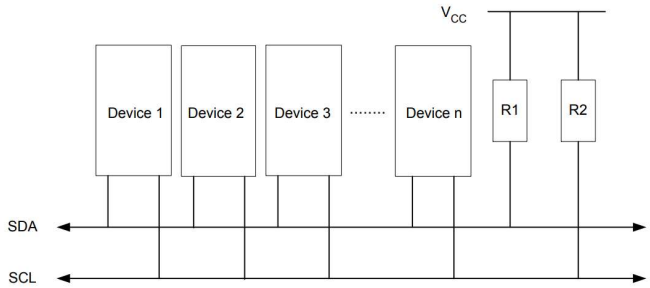
Komunikacija

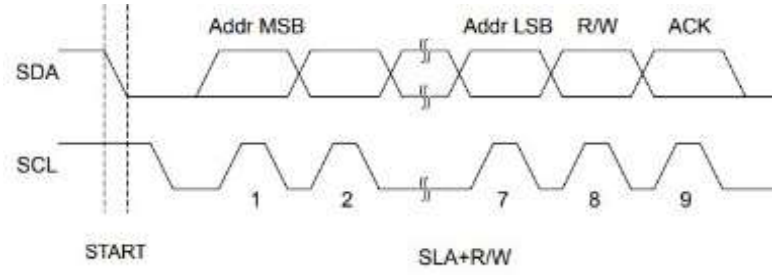
- **Serial Peripheral Interface** Bus (**SPI**)
  - sinkroni serijski protokol između SPI master i slave uređaja
  - full duplex – istovremeni prijenos podataka u oba smjera
  - tipična brzina 10 Mbps
  - povezivanje brzih vanjskih jedinica – ADC, DAC, senzori, RF moduli…
  - linije:
    - **SS** – Slave Select – master postavlja u „0“ da odabere SPI slave uređaj
    - **MOSI** – Master Output, Slave Input
    - **MISO** – Master Input, Slave Output
    - **SCK** – takt (clock) vremenskog vođenja komunikacije – sinkrono



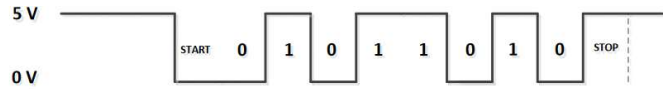
- **Inter-Integrated Circuit** (**I²C**) ili ekvivalentni Two-Wire Interface (TWI)
  - sinkroni protokol između više master i više slave uređaja
  - tipična brzina 100 kbps (standard) ili 400 kbps (fast)
  - povezivanje sporijih vanjskih jedinica – EEPROM, RTC, temperaturni senzori...
  - linije: **SCL** – clock, **SDA** – data
    - START: master stvara prijelaz SDA iz „1“ u „0“ dok je SCL=„1“, zauzima sabirnicu
    - slijedi 7-bitna adresa slave uređaja kojem je podatak namijenjen
    - adresirani slave šalje potvrdu (ACK) postavljanjem SDA u „0“ za vrijeme jednog takta
    - ovisno o odabranom smjeru komunikacije (Data Direction bit), master ili slave šalju 8-bitni podatak
    - uređaj koji prima podatak mora poslati ACK
    - STOP: master stvara prijelaz SDA iz „0“ u „1“ dok je SCL=„1“ i sabirnica je oslobođena
    - potreban vanjski pull-up otpornik jer više mastera može zatražiti sabirnicu istovremeno

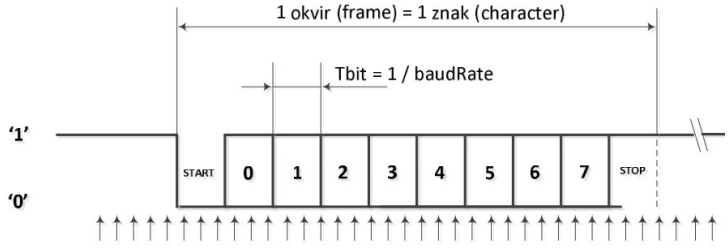
    SMBus – podskup protokola korišten na matičnim pločama računala

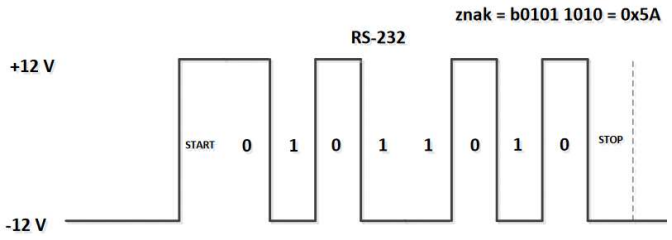




- **Universal Synchronous/Asynchronous Receiver/Transmitter** (**USART**)
  - najčešće se koristi samo asinkrona varijanta protokola (**UART**) uz prilagođene naponske razine 3.3-5V
  - starija računala su ga koristila za povezivanje periferije fizičkim RS-232 standardom ±3-15V uz neg. logiku
  - tipična brzina 9600 ili 115200 bps – unaprijed poznata na obje strane
  - danas: povezivanje mikrokontrolera i računala preko USB/UART mostova (FTDI FT232...)
  - linije: **GND**, **TX** – slanje, **RX** – prijem
    - start bit „1“ → podatkovni bitovi → stop bit „0“
    - uzorkovanje 3 puta po bitu







- Universal Serial Bus (**USB**)
  - USB 1.1 (1.5 Mbps), USB 2.0 (12 Mbps Full speed, 480 Mbps High speed), USB 3.0 (4.8 Gbps Super speed)
  - uloge: USB device ili host
    - On-The-Go: mogućnost dinamičkog biranja uloge ovisno o povezanom uređaju (za mobitele i tablete)
  - vrste prijenosa: control, bulk, interrupt, isochronous

2. Cortex-M procesorska arhitektura

Pregled ARM arhitektura

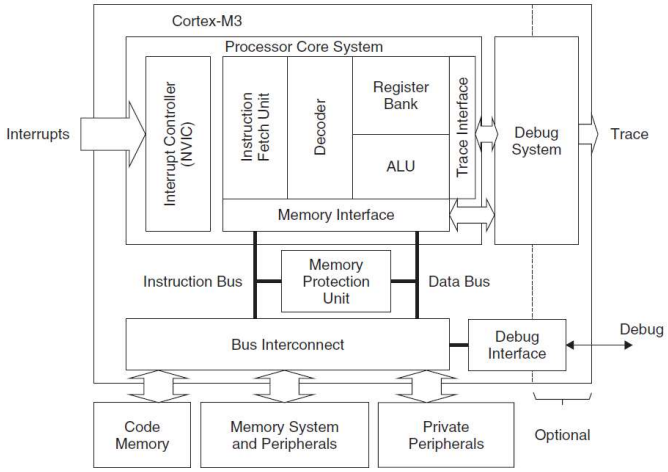
- ARM procesori (Advanced RISC Machines Ltd., poslovni model kompanije: intellectual property licensing)

- starija nomenklatura: procesor opisan brojem i sufiksom
  - ARM7TDMI: T – thumb instrukcije, D – JTAG debugging, M – brzo množilo, I – embedded ICE module
  - S – synthesizable, J – Jazelle (Java)

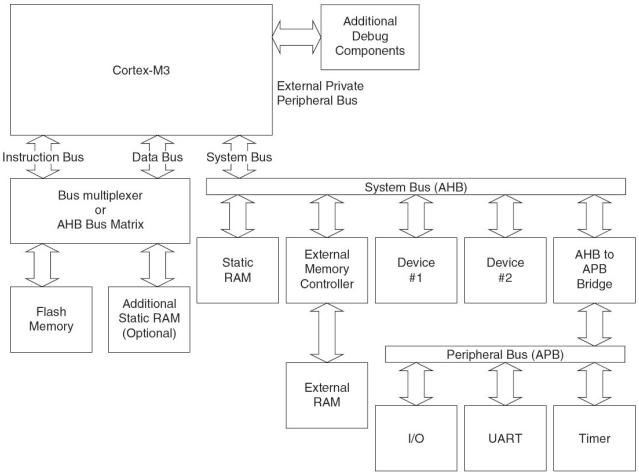
- **ARMv7** – **Cortex** – 32-bitni procesor, profili:
  - **„Application“** (A) – općenite aplikacije, visoke performanse, virtualna memorija: pametni telefoni, laptopi…
  - **„Real-Time“** (R) – ugradbene aplikacije, visoke performanse, rad u stvarnom vremenu
  - **„Microcontroller“** (M) – mikrokontrolerske aplikacije, balans perfomanse, cijele, latencije, jednostavnosti
    - jeftin µC sustav s jako niskom latencijom prekida – industrija, komunikacije, potrošna elektronika

- **ARMv8**
  - proširenje na 64-bitnu arhitekturu (AArch64)
  - ARMv8-M: dodatno TrustZone technology (sigurnost i izolacija), Helium technology (machine learning)

Cortex-M3



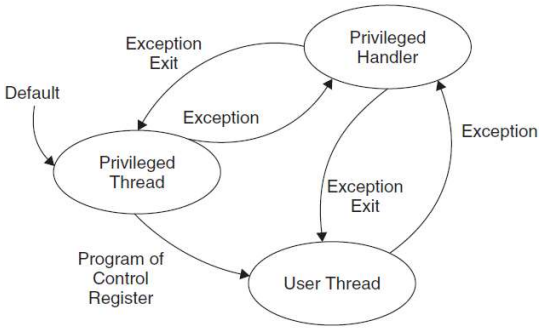
Cortex-M3 sabirničko sučelje



Cortex-M procesor

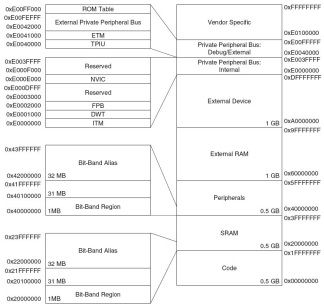
- registri:
  - `R0-R12` – registri opće namjene
  - `R13` – Stack Pointer (full-descending)
    - dekrement prije dodavanja podatka
    - `MSP` – Main SP za jezgru / privileged mode
    - `PSP` – Process SP za korisnički kod
  - `R14` – Link Register (`LR`)
  - `R15` – Program Counter (`PC`)
  - `xPSR` – Program Status Registers
    - `APSR` – Application (ALU) – NZCVQ
    - `IPSR` – Interrupt – ISR Number
    - `EPSR` – Execution – ICI/IT, Thumb state
  - `CONTROL`
    - odabir razine pristupa: **user** ili **privileged**
    - odabir SP: MSP ili PSP, samo u Thread mode

- načini rada:
  - **Thread mode** – user ili privileged, main program
  - **Handler mode** – samo privileged, exception
  - procesor se boota u privileged Thread mode
  - ključno za realizaciju OS-a



- instrukcijski set:
  - ARM instrukcije (32-bit)
  - Thumb instrukcije (16-bit) – podskup ARM seta, gušće pakiranje koda
    - potrebna dinamička zamjena ARM/Thumb stanja (problem: kašnjenje radi zamjene konteksta)
  - **Thumb-2** – nadogradnja Thumb seta uz dodatak 32-bitnih instrukcija
    - istovremeno postizanje visoke gustoće koda i složenih operacija bez potrebe za zamjenom stanja
  - Cortex-M podržava samo Thumb-2

Cortex-M memorija



- memorijska mapa: **memory-mapped I/O**
  - `0x00000000–0x1FFFFFFF` – **Code**+Data (Flash)
  - `0x20000000–0x3FFFFFFF` – **SRAM**
  - `0x40000000–0x5FFFFFFF` – **Peripherals**
  - `0x60000000–0x9FFFFFFF` – External RAM
  - `0xA0000000–0xDFFFFFFF` – External Device (I/O)
  - `0xE0000000–0xFFFFFFFF` – Private Peripherals, vendor-specific

- sabirničko sučelje: **Harvardska arhitektura**
  - **Instruction Bus** (I-Code) – AHB-Lite protokol, dohvat instrukcija iz Code regije
  - **Data Bus** (D-Code) – AHB-Lite protokol, dohvat podataka iz Code regije
  - **System Bus** – AHB-Lite protokol, dohvat instrukcija i podataka iz ostatka memorijskog prostora
    - periferne jedinice: APB protokol
  - External Private Peripheral Bus – AHB protokol, debug interface
  - moguć istovremeni dohvat sa različitih sabirnica (npr. instrukcija i podataka)
  - podržani neporavnati (unaligned) transferi u memoriji, ali nisu preporučeni zbog performansi

- **bit-banding**
  - rad s bitovima u samo jednoj load/store operaciji pomoću aliasa u kojem svakih 4 bytes predstavlja bit
  - primjena: atomarna operacija nad pojedinačnim bitovima, brzina, manje zauzeće memorije (bool type)
  - SRAM: za regiju `0x20000000–0x200FFFFF`, alias `0x22000000–0x23FFFFFF`
  - Peripherals: za regiju `0x40000000–0x400FFFFF`, alias `0x42000000–0x43FFFFFF`
  - `#define DEVICE_REG0_BIT1 ((volatile unsigned long *) 0x42000004)`

- **Memory Protection Unit** (MPU): ključna za realizaciju RTOS-a
  - onemogućavanje korisničkom zadatku pristup memoriji OS-a i drugih zadataka
  - zaštita podataka definiranjem read-only segmenata, detekcija pogrešaka (npr. corrupted stack)

Cortex-M prekidi i iznimke

- prekidi i iznimke:
  - **Reset** – kod koji se prvi izvršava nakon uključenja sustava (ili zatraženog reseta)
  - **NMI** – Non-Maskable Interrupt – kritične, nezanemarive pogreške

  - Fault iznimke:
    - **MemManage** – MPU iznimke – pristup nedefiniranoj regiji, pisanje u read-only regiju…
    - **BusFault** – pogreška na AHB sabirnici – pristup fizički nepostojećoj memoriji, uređaj nije spreman…
    - **UsageFault** – invalid opcode, pokušaj prebacivanja u ARM instruction set, dijeljenje s nulom…
    - **HardFault** – kod neke od prethodnih iznimki, pogreška pri pokretanju odgovarajuće ESR
    - Fault handler može: Reset sustava, Recovery (npr. kod nepostojećeg koprocesora), Task termination

  - zahtjevi za servisom OS-a:
    - **SVCall** – pozivanje funkcije OS-a u privilegiranom načinu rada – instrukcija SVC („blocking“ API)
    - **PendSV** – low-priority „odgođeni“ poziv, za context-switch u task scheduleru OS-a
  - **SYSTICK** – 24-bitno countdown brojilo s prekidom, za context-switch u task scheduleru OS-a

- **vanjski prekidi**: `IRQ0`...`N` (ukupno najviše 240 - veliki broj prekidnih izvora)
  - mogu se pokrenuti softverski – upisivanje broja prekida u `STIR` (jer `SETPENDx` nije dostupan)

- **Nested Vectored Interrupt Controller** (**NVIC**)
  - sklopovska prekidna arhitektura – implementacija prioriteta, gniježđenja i vektora prekida (nisko kašnjenje)
  - svaki vanjski prekid ima svoj skup registara
    - omogućavanje: `SETENAx` i `CLRENAx`
    - pending status: `SETPENDx` i `CLRPENDx`
    - prioritet: `PRI_x`
    - aktivni status: `ACTIVEx`
    - za ostale sistemske iznimke: `SHCSR` i `ICSR`
  - Interrupt Mask Registers (preko `MSR` instrukcije)
    - `PRIMASK` – mask sve osim NMI i HardFault
    - `FAULTMASK` – mask sve osim NMI
    - `BASEPRI` – mask sve ispod određenog prioriteta

- **tablica prekidnih vektora**
  - NVIC obavlja bezuvjetni skok u **Interrupt/Exception Service Routine** (ISR/ESR) na temelju adrese u tablici
  - na početku memorije `0x00000000`: prvo inicijalna vrijednost `MSP`, pa onda Reset vector, NMI vector...
  - ako je potrebno mijenjati vektore, relokacija npr. u SRAM

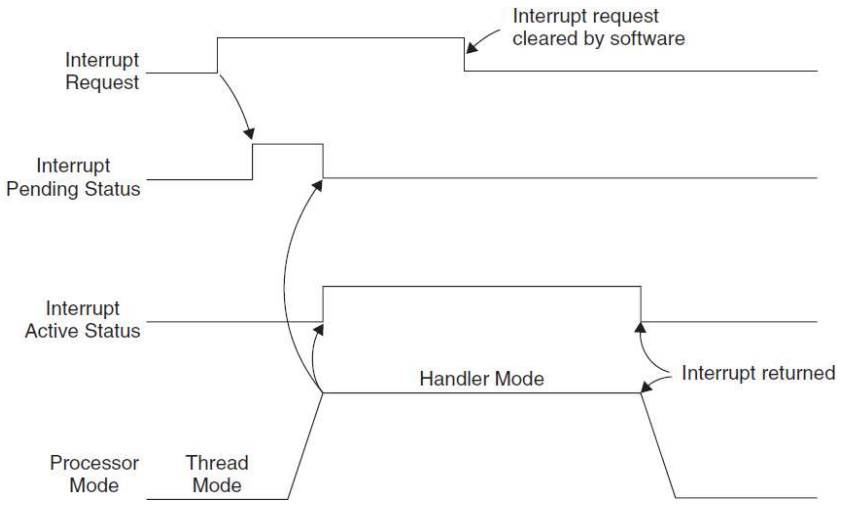
- **Priority Level** (`PRI_x`) registar za programabilne iznimke:
  - **preempt priority** (istiskivanje) – samo strogo veći preempt priority može prekinuti trenutnu iznimku
  - **subpriority** – određuje prioritet unutar preempt skupine
  - `PRIGROUP` bitovi `AIRCR` registra određuju podjelu Priority Level registra na preempt i subprio po bitovima
  - što manja vrijednost↘, to veći prioritet↗
    - sve programabilne iznimke su nižeg prioriteta od Reset (-3), NMI (-2), HardFault (-1)
    - FIQ ne postoji (za razliku od stare ARM arhitekture)
  - primjer
    - 8 razina prioriteta (3-bitno)
    - implementirani bitovi 7:5
    - neimplementirani bitovi 4:0
    - vrijednosti prioriteta 0x00, 0x20, 0x40...
    - za PRIGROUP=7 svi bitovi su subprio
    - za PRIGROUP=6 najviši bit postaje preempt
    - za **PRIGROUP=5** dva najviša bita su preempt

Priority Group		Preempt Priority Field	Subpriority Field
0	xxxxxxxy	Bit [7:1]	Bit [0]
1	xxxxxxyy	Bit [7:2]	Bit [1:0]
2	xxxxxyyy	Bit [7:3]	Bit [2:0]
3	xxxxyyyy	Bit [7:4]	Bit [3:0]
4	xxxyyyyy	Bit [7:5]	Bit [4:0]
5	xxyyyyyy	Bit [7:6]	Bit [5:0]
6	xyyyyyyy	Bit [7]	Bit [6:0]
7	yyyyyyyy	None	Bit [7:0]

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Preempt priority		Sub priority					

- **gniježđenje** (**nested**)
  - automatsko prekidanje trenutne iznimke iznimkom višeg prioriteta
  - iznimke istog ili nižeg prioriteta od trenutne su blokirane (nedozvoljen re-entrant)
  - prekinuti prekidi su i dalje aktivni

- proces obrade prekida:
  - kada se pojavi zahtjev za prekidom (IRQ), NVIC postavlja **interrupt pending** status
    - pending ostaje aktivan čak i ako se IRQ deaktivira prije obrade prekida
  - odustajanje od obrade prekida – ručnim brisanjem pending statusa
  - **prihvaćanje** – procesor ulazi u ISR/ESR (prebacuje se u Handler mode)
    - istovremeno, automatska pohrana konteksta „**stacking**“ (`R0`-`R3`, `R12`, `LR`, `xPSR` i `PC` na stog)
    - automatski resetira pending status i postavlja aktivni status
    - ISR/ESR mogu biti u potpunosti programirane u C-u zbog stackinga i sklopovskog gniježđenja
  - pri izlasku iz ISR/ESR, vraćanje vrijednosti registara sa stoga „**unstacking**“
  - **latencija** – vrijeme od IRQ zahtjeva do početka izvođenja ISR – ~12 ciklusa
    - dugo trajanje izvođenja ISR višeg prioriteta može dodatno povećati kašnjenje

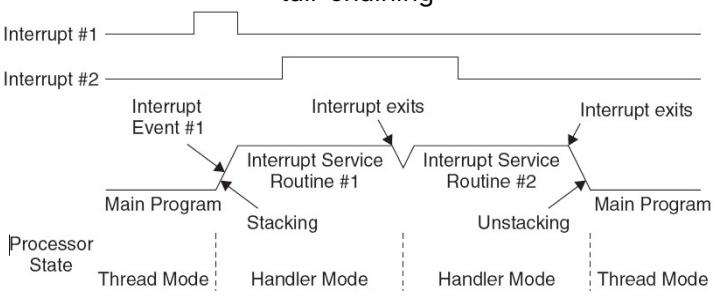


- rubni slučajevi:
  - za stalno aktivan IRQ – nakon povratka iz ISR, pending status je opet postavljen, ponovno ulazi u ISR
  - višestruko ponovljeni IRQ prije ulaska u ISR/ESR – rezultira samo jednim pending zahtjevom

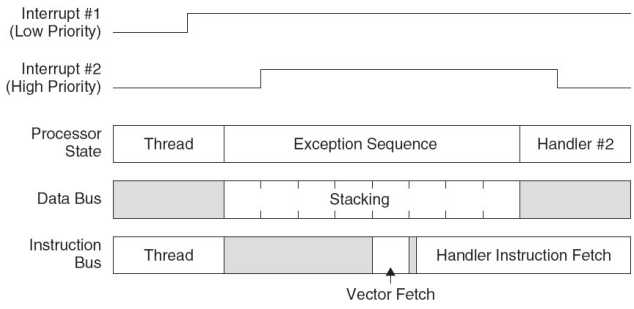
- priprema sustava za korištenje iznimki nakon reseta:
  - podešavanje stoga, podešavanje tablice vektora, podešavanje prioriteta prekida, i omogućavanje prekida
  - prije omogućavanja prekida poželjno je resetirati pending status
    - pending se događa i kada je prekid onemogućen – omogućavanje prekida tada može odmah ući u ISR
    - tranzijentne pojave prilikom uključenja sustava mogu uzrokovati lažni IRQ i pending status

- optimizacije kod obrade više prekida:
  - **tail-chaining** – kada manje prioritetan prekid stigne tijekom ISR, prelazak direktno u niži bez unstackinga
  - **late arrival** – kada više prioritetan prekid stigne tijekom stackinga, prelazak direktno u viši bez nižeg

tail-chaining



late arrival



- upravljanje potrošnjom pomoću prekida:
  - načini rada niske potrošnje: **Sleep** i **Deep sleep** (točno ponašanje ovisi o implementaciji)
  - odabir moda: `SLEEPDEEP` bit u System Control registru
  - instrukcija `WFI` (wait for interrupt) – čekanje na prekid
  - instrukcija `WFE` (wait for event) – čekanje na prekid ili vanjski signal događaja (`RXEV`)
  - SleepOnExit – mogućnost automatskog vraćanja procesora u stanje niske potrošnje nakon obrade iznimke

3. Oblikovanje programske potpore za ugradbene računalne sustave

Sustav za rad u stvarnom vremenu

- specifičnosti programske potpore za ugradbene računalne sustave:
  - rad u stvarnom vremenu: pravodobnost, predvidljivost, otpornost na pogreške, dizajn za max. opterećenje
  - pouzdanost, sigurnost, održivost, raspoloživost, učinkovitost: (veličina koda, performanse, energetski)
  - izravan pristup sklopovlju, paralelizam izvršavanja zadataka...

- sustav za rad u stvarnom vremenu:
  - **deadline** – logički ispravan, deterministički garantiran vremenski odziv vanjske događaje
  - vremenski okviri: worst-case ($WCET$) i best-case ($BCET$) execution time
  - hard real-time
    - neispunjen deadline uzrokuje kritičan pad sustava
    - detekcija kritičnih događaja, sustavi upravljanja, kritični sustavi, obrambeni sustavi...
  - soft real-time
    - neispunjen deadline uzrokuje lošije performanse, ali ne i kritičnu pogrešku
    - čitanje podataka s tipkovnice, grafički prikaz zaslona ili konzole...
  - firm real-time
    - nekoliko neispunjenih deadline je ok, ali previše može uzrokovati kritičnu pogrešku
    - auto navigacija...

Česti problemi pri oblikovanju programske potpore

- problem dijeljenih resursa:
  - prekid može promijeniti podatke tijekom rada s tim podacima, uzrokujući nestabilnosti u sustavu
  - **atomarna operacija** – dio programa koji se izvodi kao cjelina, biti prekinuta
  - **kritična sekcija** – skup instrukcija koje obavljaju atomarnu operaciju
    - ostvaruje se omotavanjem koda ulaskom `_DISABLE_INTR()` i izlaskom `_ENABLE_INTR()`
    - povećava latenciju prekida radi privremenog onemogućenja prekida za vremenski kritične prekide
    - pripaziti na situacije gniježđenja kritičnih sekcija

- ključna riječ `volatile`:
  - govori compileru da ne smije raditi pretpostavke (optimizacije) o sadržaju memorijske adrese
  - compiler onda ne optimira čitanje i pisanje, nego ih izvršava odmah
  - za rad sa sklopovskim registrima, situacije gdje drugi task ili prekidna rutina može promijeniti varijablu

- tijek podataka (streaming):
  - procesiranje podataka koji kontinuirano, neprekidno pristižu – obrada signala, komunikacije...
  - struktura podataka **red** (queue, FIFO) – najčešće statički implementirana (**circular buffer**)

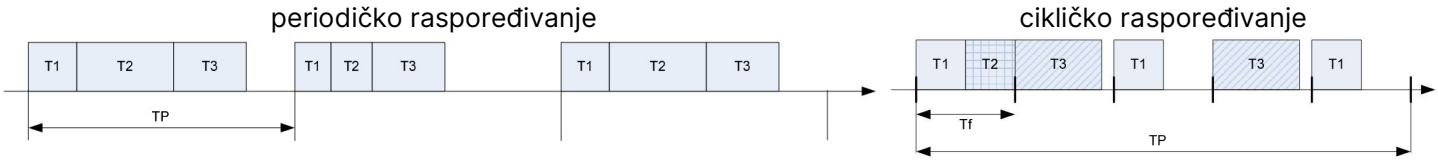
Ostvarivanje višezadaćnosti bez operacijskog sustava

- zadatak (ili proces)
  - funkcija (programska cjelina) koja se izvodi sekvencijalno u beskonačnoj petlji
  - „dispatching" – alokacija CPU resursa određenom zadatku
  - **periodički** zadatci – moraju se izvršavati u redovnim intervalima
  - **aperiodički** zadatci – nemaju obrazac izvršavanja, mogu biti potaknuti vanjskim događajima

- klasifikacija algoritama raspoređivanja zadataka:
  - **preemptive** – zadatak može biti prekinut za vrijeme izvođenja
  - **non-preemptive** – zadatak se mora izvršiti do kraja prije nego se započne izvođenje novog
  - **static** (off-line) – raspoređivanje na temelju fiksnih parametara prije izvođenja
  - **dynamic** (on-line) – raspoređivanje temeljeno na ponašanju zadataka za vrijeme izvođenja
  - **optimal** – optimalan po zadanom kriteriju
  - **heuristic** – teži optimalnom, ali ne garantira

- prekidne rutine (ISR)
  - služe za brzu obradu zahtjeva za prekid
  - pristup ulazno-izlaznim jedinicama – pripremaju podatke za zadatke

- vremensko upravljanje zadacima (time-triggered approach)
  - implementirano pomoću timera (prekida) s unaprijed zadanim statičkim rasporedom zadataka
  - interakcija s ulazno-izlaznim jedinicama je strogo polling (bez prekida)
  - **periodički** (periodic)
    - svi zadatci se redom izvode konstantnom učestalošću (predefinirani period izvođenja prvog zadatka…)
    - uvjet: zbroj worst-case execution time svih zadataka mora biti kraći od perioda
  - **ciklički** (cyclic executive)
    - slično periodičkom, ali dodatno dijelimo period dijelimo na vremenske okvire



- **non-preemptive event-triggered scheduling** (zasnovano na događajima bez istiskivanja)
  - glavna upravljačka petlja izvodi stalnu iteraciju po svim zadatcima
  - **Round-Robin** – iteracija u predefiniranom redoslijedu uz polling
    - with Interrupts – prekidne rutine postavljaju zastavice, provjeravanje i zadatak u glavnoj petlji
    - ako je potrebna brza reakcija, dio koda može se prebaciti u prekidnu rutinu
  - **Function Queue Scheduling**
    - prekidne rutine stavljaju funkcije u (prioritetni) red, prozivanje iz reda u glavnoj petlji
    - nema istiskivanja – nadolazeći zadatak višeg prioriteta mora čekati da se trenutni dovrši
    - izgladnjivanje –zadatci najnižeg prioriteta nikada ne dolaze na red

```
typedef void (*FuncPtr)();
Queue<FuncPtr> tasks;

void DEVICE_A_HANDLER() { /* ... */ }

void interrupt deviceA_ISR() {
    tasks.Enqueue(&DEVICE_A_HANDLER);
}
```

```
void main() {
    while (1) {
        if (!tasks.empty()) {
            FuncPtr task = tasks.Dequeue();
            (*task)();
        }
    }
}
```

- **non-preemptive cooperative multitasking** (kooperativna višezadaćnost bez istiskivanja)
  - „coroutines“ – paralelni zadatci dobrovoljno prepuštaju kontrolu (ne mogu se međusobno prekinuti)
  - implementacija pomoću finite state machine (FSM), komunikacija putem globalnih varijabli

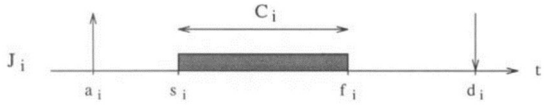
- problem: ne postoji mehanizam prekidanja tekućeg zadatka i prepuštanja tijeka zadatku višeg prioriteta
- **preemptive multitasking** (**višezadaćnost s istiskivanjem**)
  - zahtjeva jezgru operacijskog sustava s raspoređivačem zadataka (task scheduler)
  - najmoćnija implementacija, jedina sposobna za rad s hard real-time zahtjevima
  - OS odlučuje kada dolazi do zamjene konteksta, i koji će se zadatak izvesti nakon zamjene

Ostvarivanje višezadaćnosti uz operacijski sustav za rad u stvarnom vremenu (RTOS)

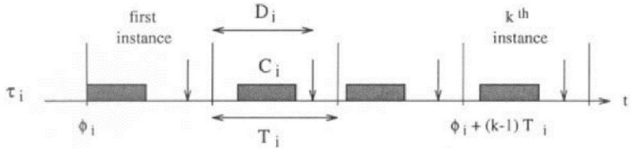
- RTOS pruža API za višezadaćnost:
  - apstrakcija zadataka: task funkcije, prioriteti, stanje stoga, zamjena konteksta…
    - mogućnost **prekidanja zadataka nižeg prioriteta** od strane zadataka višeg prioriteta
    - redoslijed izvođenja zadataka: task scheduler
  - komunikacija između zadataka: sinkronizacijski mehanizmi
    - signaliziranje (semafori), kritične sekcije, redovi poruka…
    - zaštita memorije procesa/zadatka
  - osnovna struktura programa: glavna funkcija inicijalizira RTOS i pokreće zadatke s željenim prioritetom

- **task scheduling** – raspoređivanje zadataka:
  - skup n zadataka: $J = \{J_1, J_2, \dots, J_n\}$
  - **raspored** (schedule) – funkcija $\sigma(t)$
    - $\sigma(t) = i$ za t u kojem se izvršava zadatak J_i, $\sigma(t) = 0$ ukoliko je procesor idle (ne izvršava zadatke)
    - **promjena konteksta** – trenutak u kojem funkcija $\sigma(t)$ mijenja vrijednost
    - **istiskivanje** (preemptive) – promjena konteksta je moguća u bilo kojem trenutku izvođenja zadatka
  - algoritmi raspoređivanja aperiodičkih zadataka: EDD, EDF, Tree search, LDF, EDF*, Spring
  - algoritmi raspoređivanja periodičkih zadataka: RMS, DMS, EDF
  - U – **total processor utilization** (faktor opterećenja procesora)
    - ukupna vremenska iskorištenost procesora od strane skupa zadataka
    - idle time: $1 - U$

- vremenski parametri zadatka J_i:
  - a_i – arrival time (vrijeme dolaska) – trenutak dolaska zahtjeva
  - d_i – deadline (rok izvršavanja) – vremenski period u kojem zadatak mora biti obavljen
  - s_i – start time (vrijeme početka izvršavanja) – vrijeme u kojem počinje izvršavanje zadatka (latencija)
  - f_i – finish time (vrijeme završetka izvršavanja) – za hard real-time, mora biti prije deadline-a
  - C_i – computation time (vrijeme izračuna) – obavljanje posla
  - $C_i = f_i - s_i$

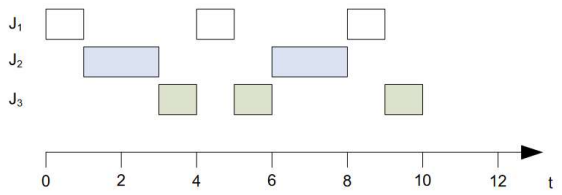


- periodički zadatci:
  - T_i – period ponavljanja i-tog zadatka
  - Φ_i – faza – vrijeme pojave prve instance i-tog zadatka
  - $r_{i,k}$ – release time – vrijeme pojave k-te instance i-tog zadatka
  - D_i – relative deadline – vrijeme u kojem i-ti zadatak nakon pojave treba biti izvršen
  - $d_{i,k}$ – absolute deadline – apsolutni rok za završetak i-tog zadatka
  - $r_{i,k} = \Phi_i + (k-1)T_i$
  - $d_{i,k} = \Phi_i + (k-1)T_i + D_i$
  - $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$



- **Rate Monotonic Scheduling** (**RMS**)
  - **optimalan algoritam za statičke prioritete**
    - ostali statički algoritam mogu biti jednaki, ali ne i bolji od RMS
  - pretpostavke:
    - $C_i = WCET$ – poznato je vrijeme izvođenja u najgorem slučaju
    - $\boldsymbol{D_i = T_i} \rightarrow d_{i,k} = \Phi_i + kT_i$ (zadatak mora biti završen do sljedećeg perioda)
    - vrijeme zamjene konteksta je zanemarivo
  - algoritam:
    - svakom zadatku J_i unaprijed je poznat period ponavljanja T_i
    - svakom zadatku J_i se **unaprijed statički dodjeljuje viši prioritet onom zadatku s manjim periodom**
    - **preemptive**: zadatci višeg prioriteta mogu prekidati zadatke nižeg prioriteta
  - kritični trenutak – najgori mogući trenutak pojavljivanja zadatka, najdulje vrijeme odgovora
    - to je trenutak u kojem se istovremeno pojavljuju i svi zadaci višeg prioriteta
  - **dovoljan uvjet uspješne rasporedivosti** (nije nužan):
    - $U \leq n\left(2^{\frac{1}{n}} - 1\right)$
    - uvjet garantira da u kritičnom trenutku neće doći do nemogućnosti odgovaranja na vrijeme
    - teži asimptomatskoj granici: $U_{lb} = \lim_{n \to \infty} n\left(2^{\frac{1}{n}} - 1\right) = \ln 2 \approx 0.693$

Proces	Vrijeme izvođenja (C_i)	Period ponavljanja (T_i)
J1	1	4
J2	2	6
J3	3	12

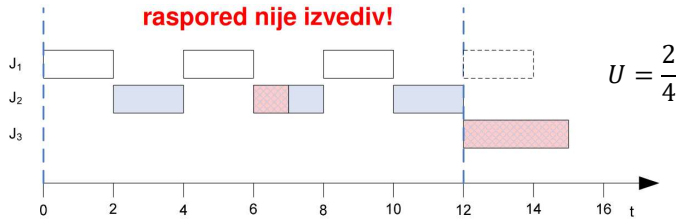


$$n = 3$$

$$U = \frac{1}{4} + \frac{2}{6} + \frac{3}{12} = \frac{5}{6} \approx 0.833$$

$$3\left(2^{\frac{1}{3}} - 1\right) \approx 0.779$$

Proces	Vrijeme izvođenja (C_i)	Period ponavljanja (T_i)
J1	2	4
J2	3	6
J3	3	12



$$U = \frac{2}{4} + \frac{3}{6} + \frac{3}{12} = \frac{5}{6} = 1.25$$

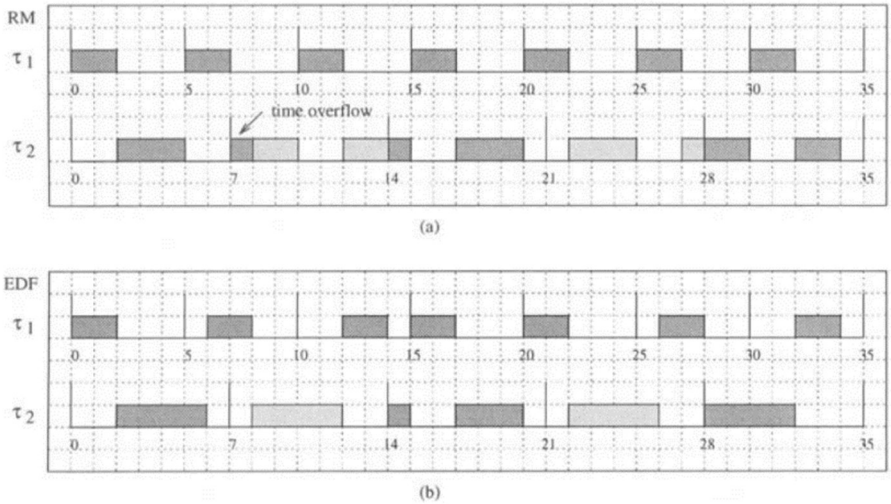
- **Earliest Deadline First** (**EDF**)
  - **optimalan algoritam za dinamičke proritete**
    - ako postoji izvedivi raspored uz dinamičke prioritete zadataka, EDF će ga sigurno pronaći
  - pretpostavke:
    - $C_i = WCET$ – poznato je vrijeme izvođenja u najgorem slučaju
    - u svakom trenutku, poznato je vrijeme preostalo do apsolutnog deadline-a $d_{i,k}$
    - $\boldsymbol{D_i \leq T_i}$ (zadatak mora biti završen do sljedećeg perioda ili brže)
  - algoritam:
    - **najviši prioritet dinamički se dodjeljuje zadatku s najbližim deadline-om** u svakom trenutku
    - trenutni zadatak J_i prekida se čim se pojavi zadatak s bližim deadline-om
  - **nužan i dovoljan uvjet uspješne rasporedivosti**:
    - $U \leq 1$
  - problem: složena praktična implementacija

Proces	Vrijeme izvođenja (C_i)	Period ponavljanja (T_i)
J1	2	5
J2	4	7

$$n = 2$$

$$U = \frac{2}{5} + \frac{4}{7} \approx 0.971$$

$$2\left(2^{\frac{1}{2}} - 1\right) \approx 0.828$$



4. Operacijski sustavi za rad u stvarnom vremenu

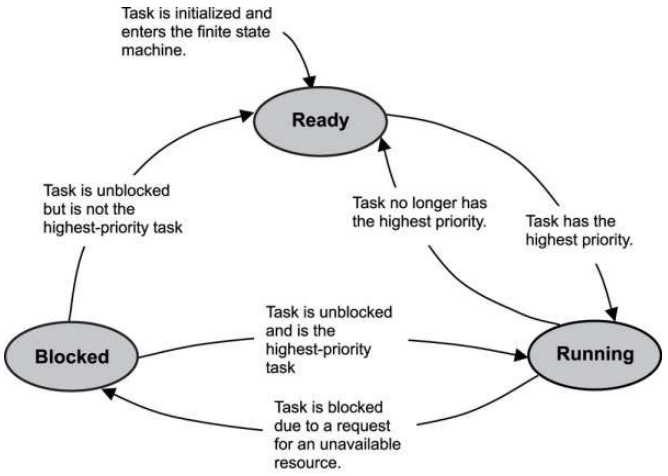
Osnovni dijelovi jezgre RTOS-a

- operacijski sustavi za rad u stvarnom vremenu (RTOS)
  - programsko okruženje koje omogućuje raspoređivanje zadataka s vremenskim ograničenjima
  - temelj za razvoj aplikacijskog koda i upravljanje resursima sustava
  - **kernel** (jezgra) – minimalna implementacija OS-a
    - task scheduler
    - jezgrini objekti: zadatci, semafori, redovi poruka…
    - servisi: vremenski servisi, prekidi, upravljanje resursima…
  - neki RTOS-ovi: VxWorks, VRTX, pSOS, Nucleus, QNX, AMX, FreeRTOS

- usporedba s GPOS (Windows, Unix):
  - sličnosti: podrška za višezadaćnost, upravljanje resursima sustava, OS service API, hardware abstraction
  - razlike: RTOS je pouzdan, radi u stvarnom vremenu, portabilan, zahtjeva manje memorije i pohrane

- **task scheduler**
  - omogućuje jednoprocesorski multitasking – odlučuje kada se izvodi i prekida svaki zadatak
  - schedulable entity – objekt koji se može natjecati za CPU vrijeme (zadatak i proces)
  - dispatcher – dio schedulera koji obavlja zamjenu konteksta, promjenu trenutnog zadatka

- **zadatak** (task)
  - programska cjelina koja se može raspoređivati, tipično funkcija s beskonačnom petljom
  - **task control block** (TCB) u linked listi: id/naziv, prioritet, stanje, kontekst (registri + stog + PC), extra data
  - stanja (implementirana po modelu finite state machine):
    - **pripravan** (ready) – spreman za izvršavanje, ali čeka CPU resurse od schedulera
    - **blokiran** (blocked) – neaktivan, čeka na vanjski događaj, resurs ili vremenski timer da se aktivira
    - **u izvođenju** (running) – izvodi se na CPU dok ga ne prekine scheduler, ili se dobrovoljno blokira (čeka)
    - ostala opcionalna stanja: suspended, pended, delayed…
  - intertask primitives (sinkronizacija i komunikacija između zadataka)
    - semafori, redovi poruka, signali, cjevovodi…
    - česta upotreba: realiziranje **kritičnih sekcija** za izbjegavanje problema dijeljenih resursa



- **proces** (slično kao i zadatak)
  - bolje međusobno razdvajanje, zaštita memorije (virtualna memorija)
  - **thread** – neovisni slijed izvođenja unutar jednog procesa (multithreading)

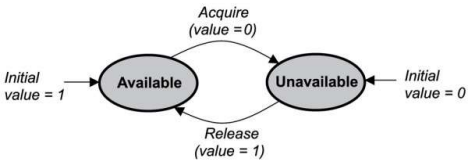
- **reentrant funkcija** (thread-safe)
  - funkcija koja se može pozvati od strane više zadataka istovremeno i pritom uvijek ispravno funkcionirati
  - mora rukovati s podacima i pristupati sklopovlju atomarno
  - ne smije pozivati non-reentrant funkcije

Semafori

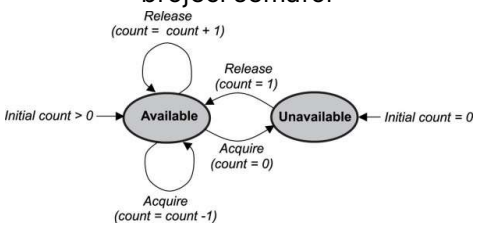
- **semafor**
  - kernel object kojeg zadaci mogu zauzeti ili otpustiti (kao flag)
  - za sinkronizaciju i međusobno isključivanje: (multiple-)wait-and-signal, (recursive) shared resource access
  - **semaphore control block** (SCB): id/naziv, vrijednost, lista zadataka na čekanju
  - API za rad s semaforima uzimaju objekt (pointer na SCB) kao parametar – broj semafora nije ograničen
  - česti problemi: inverzija prioriteta i deadlock

- vrste semafora s obzirom na tip vrijednosti:
  - **binarni** – „0“ (zauzet) ili „1“ (slobodan), ponaša se kao nezaštićeni globalni resurs bilo kojeg procesa
  - **brojeći** – „0“ (zauzet) ili „>0“ (slobodan), N puta ga može zauzeti N procesa istovremeno
  - **mutex** – zaključan ili otključan uz dodatne značajke
    - vlasništvo – samo zadatak koji je zaključao mutex može ga i otključati
    - rekurzivno zaključavanje – zadatak koji je zaključao mutex može ga višestruko zaključati (lock count)
    - sigurno brisanje zadataka – zaštita od brisanja zadatka koji drži mutex zaključanim
    - implementirani protokoli izbjegavanja problema
    - primjena: realizacija kritičnih sekcija

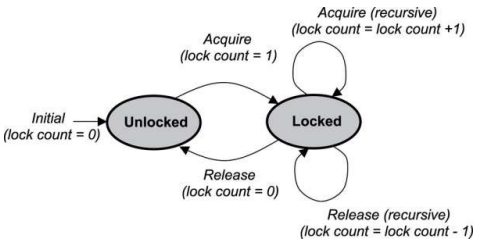
binarni semafor



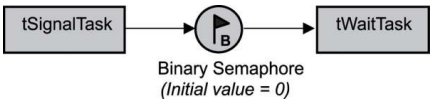
brojeći semafor



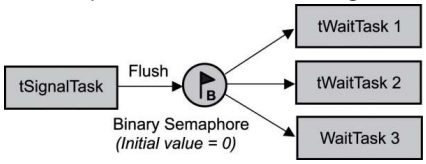
mutex



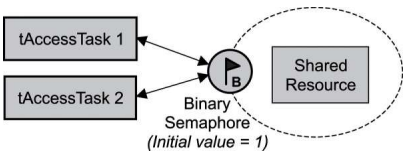
wait-and-signal



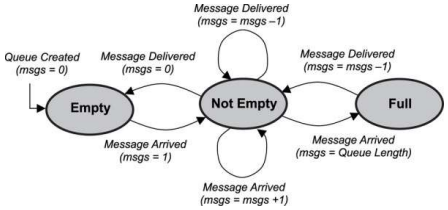
multiple-task wait-and-signal



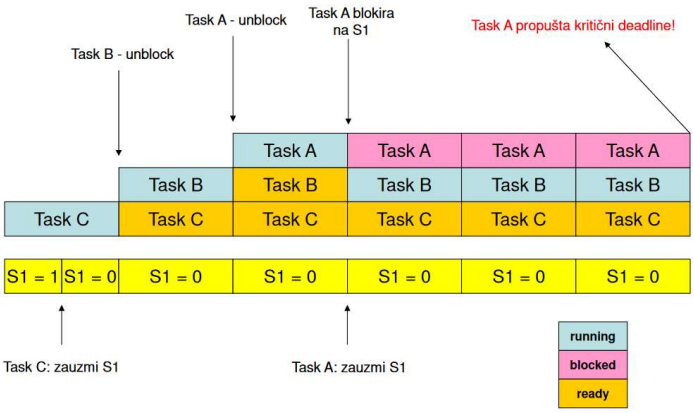
single shared-resource-access

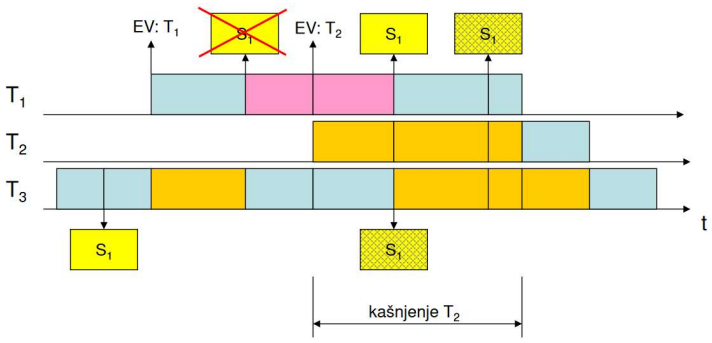


recursive shared-resource-access

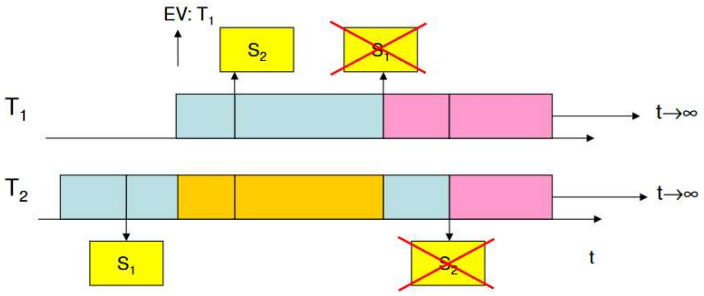


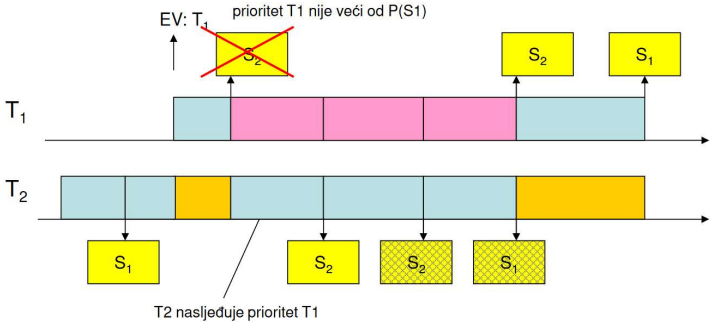
- **priority inversion** problem (inverzija prioriteta):
  - primjer: mutex i 3 zadatka višeg, srednjeg i nižeg prioriteta
    - niži drži mutex zaključanim
    - viši pristiže i zahtjeva mutex, ali se blokira jer niži prvo treba dovršiti operaciju s mutexom
    - inverzija: sustav ne izvršava najprioritetniji zadatak
    - dodatan slučaj **unbounded**: pristiže srednji zadatak, pa se niži ne izvršava dok drži mutex zaključanim
      - u tom slučaju bi niži ipak trebao imati prednost nad srednjim kako bi viši završio što ranije
  - **priority inheritance protocol** (protokol nasljeđivanja prioriteta)
    - dinamički se povećava prioritet nižeg zadatka na razinu višeg zadatka koji zahtjeva mutex
    - nakon što zadatak završi operaciju, privremeno povećanje prioriteta se poništava
    - rješava unbounded slučaj, ne uklanja inverziju u potpunosti





- **deadlock** (potpuni zastoj)
  - 2 mutexa i 2 zadatka međusobno čekaju jedno drugog da oslobode svoj mutex
  - nadogradnja: **priority ceiling protocol** (protokol stropnog prioriteta)
    - svakom mutexu dodjeljuje se stropni prioritet – najveći prioritet zadatka koji mu može pristupiti
    - viši zadatak može biti dodatno blokiran pri zahtjevu mutexa ako postoji neki drugi zaključani mutex s stropnim prioritetom većim ili jednakim prioritetu višeg zadatka





Ostali objekti jezgre

- **redovi poruka** (message queues)
  - Inter-Process Communication (IPC) – zadaci i prekidi mogu komunicirati i razmjenjivati poruke
  - red poruka (FIFO; LIFO ili prio) s jedinstvenim identifikatorom, memorijskim resursima, brojem elemenata
  - slanje i primanje poruka:
    - neblokirajuće (pogodno za ISR): kopiranje memorije od pošiljatelja u red, pa u memoriju primatelja
    - blokirajuće (s timeoutom): ako primatelj ili pošiljatelj još ne postoji, stavlja se na listu čekanja zadataka
    - pisanje u puni red ili čitanje praznog reda – pogreška ili blokiranje
    - čitanje poruke: pop ili peek
  - načini upotrebe:
    - non-interlocked one-way – slanje podataka iz ISR u zadatak bez blokiranja, fire and forget
    - interlocked one-way – potvrda primitka poruke pomoćnim semaforom, pouzdanije
    - two-way – dvosmjerno, client/server arhitektura
    - broadcast – slanje jedne poruke velikom broju zadataka primatelja