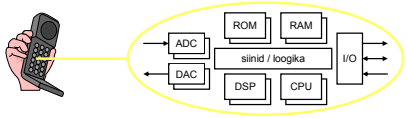
Arvutusprotsesside organiseerimine

ehk kuidas programm mällu satub

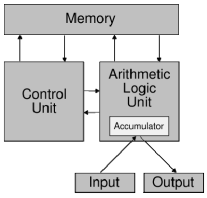
Sardsüsteem = programm + mikroskeem

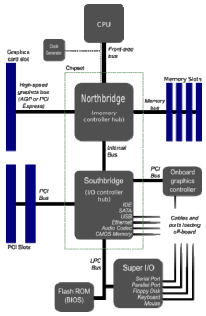
[rakendus + OS] + [digitaal + analoog]



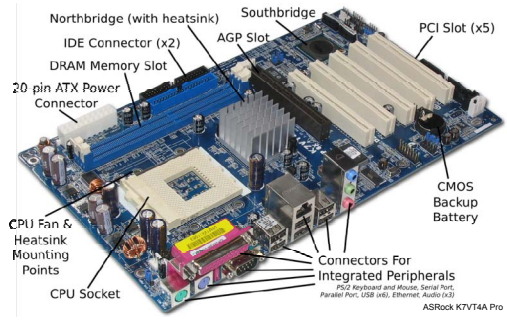
Von Neumanni arhitektuur

- Mälu
- Protsessor
  - Juhtseade
  - ALU

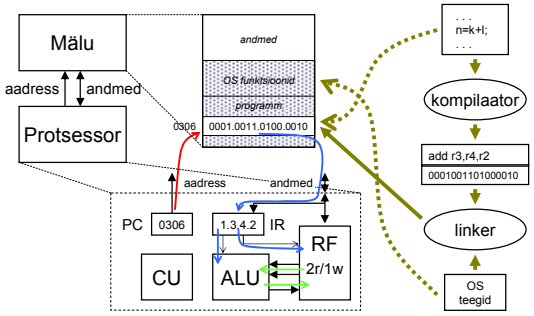




Mälu & Protsessor



Mälu & Protsessor



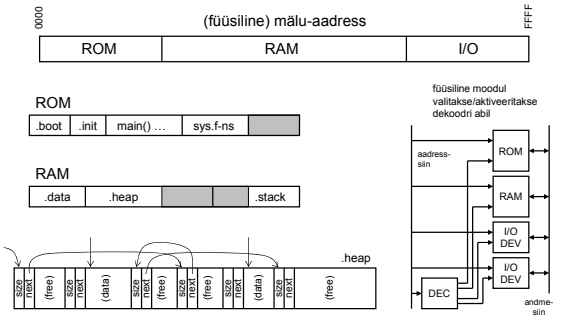
Sardtarkvara süntees

- Süsteemi analüüs
- RTOS & rakenduste valimine
- Tükeldamine
- Liideste defineerimine
- Moodulite projekteerimine
- Tarkvara integreerimine
- Tarkvara paigaldamine (ROM)
- Tarkvara kompileerimine & linkimine

Sardtarkvara

- Üks ja ainuke rakendusprogramm
  - Mälujaotus
    - programm – rakendus + süsteem
    - andmed – staatilised / dünaamilised / pinu
  - Programmi töö
    - alglaadimine
      - protsessori initsialiseerimine
      - programmi initsialiseerimine
    - programm tsüklis
    - süsteemipöördused

Mälujaotus – üks rakendus

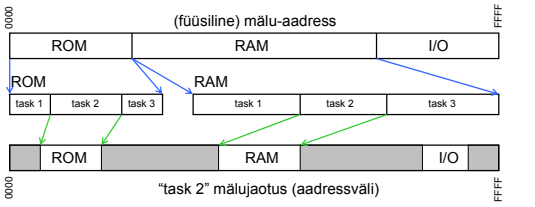


Sardtarkvara

- Mitu rakendusprogrammi (ülesannet)
  - lisaks üksikule programmile
  - Mälujaotus
    - eri programmide andmed/programm eri kohtades
    - andmete kaitsmine
      - MMU – Memory Management Unit
  - Programmide töö
    - süsteemi initsialiseerimine
    - ülesannete vahetamine
      - planeerimine, CPU sisu salvestamine, …

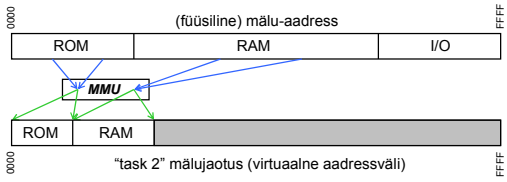
Mälujaotus – mitu rakendust

- Iga rakendusprogramm füüsiliselt erineval aadressil
  - rakenduse sisemised mälupiirkonnad samad (.data, .heap jne.)
  - MMU-d kasutatakse mälu kaitsmiseks valepöörduste eest
  - linkimine keerukas – aadressid vaja eraldi ette anda



Mälujaotus – mitu rakendust

- Kõikide rakendusprogrammide jaoks on mälujaotus ühesugune – nn. virtuaalne aadressväli (virtuaalmälu)
  - rakenduse sisemised mälupiirkonnad samad (.data, .heap jne.)
  - MMU vajalik nii mälupiirkondade kaitsmiseks kui ka virtuaalse aadressi "tõlkimisel" füüsiliseks aadressiks
  - linkimine lihtne – sama kõikidele rakendustele



Mälujaotus

- ROM
  - RO segment (mitu programmi/kasutajat)
  - *.boot* piirkond
    - CPU initsialiseerimine
    - katkestusvektorid
    - pinu jt. mälupiirkondade aadressid
      - … või teostatakse (RT)OS-i poolt
  - *.init* piirkond
    - parameetrid programmile / tulemus OS-le
    - objektide algväärtustamine (nt. C++)

Mälujaotus

- ROM (jätkub…) *.text* piirkond
  - main() …
    - rakendusprogrammid
    - kompileeritult või teekidest .o-failidena
  - sys.f-ns
    - standartsed alamprogrammid
      - kompilaatori-spetsiifilised
    - süsteemsed alamprogrammid
      - OS-spetsiifilised
      - S/V draiverid jms.

Mälujaotus

- RAM
  - RW segment (mitu programmi/kasutajat)
  - *.data* piirkond
    - staatilised (globaalsed) muutujad/andmed
  - *.heap* piirkond
    - dünaamilised muutujad/andmed
    - kasutatud/vabade plokkide nimekiri
      - prügikogumine (garbage collection) aega-ajalt vajalik
      - vajadusel suurendatakse OS-i poolt
  - *.stack* piirkond
    - lokaalsed muutujad/andmed (pinu)

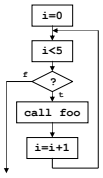
Mälukujutis

- a.out / .exe / .com
  - OS-i poolt mällu laetav programmi "toorik"
  - Sisaldab
    - programmi initsialiseerimine & sisu
    - algväärtustatud andmed (muutujad & konstandid)
      - osa globaalsetest muutujatest
  - OS-i poolt teostatakse
    - mälujaotuse ette valmistamine
      - piirkondade eraldamine (MMU), viidad registritesse, …
    - programmi juhtimine
      - alustamine, katkestamine, OS-i pöördused, …

Kompileerimine

- Etapid
  - teisendamine abstraktseks operatsioonide jadaks
  - optimeerimine
- Programmi teisendamine operatsioonide jadaks
  - Keelekonstruktsioon –> baas-plokkide jada
  - Baas-plokk ~ avaldis

```
for (i=0;i<5;i++) { foo(); }
```




```
<for_cycle> ::= 'for' '(' (<expr>)* ';' (<expr>)* ';'
                          (<expr>)* ')' [<statement>] ';'
```


Kompileerimine

- Programmi teisendamine operatsioonide jadaks
  - Baas-plokk ~ avaldis
  - Avaldis –> operatsioonide jada
    - nn. Poola kuju (Polish notation, prefix notation, Jan Łukasiewicz)
    - operatsioonide prioriteedid
    - primaased avaldised
      - sulud, muutujad, funktsioonid, …
    - pinu või virtuaal-registrid
  - Ühised alamavaldised

```
x = a + b + c ;
```

```
r1 <- + a b
r2 <- + r1 c
x <- r2
```

```
y = a + b * c ;
```

```
r1 <- * b c
r2 <- + a r1
y <- r2
```

```
z = (a + b) * c ;
```

```
r1 <- + a b
r2 <- * r1 c
z <- r2
```


Kompileerimine

- Avaldis VHDL-s (BNF, lihtsustatud)

```
<expression>          ::= <relation> [ <logic_operation> <relation> ]*
<logic_operation>     ::= 'and' | 'or' | 'xor' | 'nand' | 'nor' | 'xnor'
<relation>            ::= <shift_expression> [ <relational_operation>
                                               <shift_expression> ]
<relational_operation> ::= '=' | '/=' | '<' | '<=' | '>' | '>='
<shift_expression>    ::= <simple_expression> [ <shift_operator>
                                                <simple_expression> ]
<shift_operator>      ::= 'sll' | 'srl' | 'sla' | 'sra' | 'rol' | 'ror'
<simple_expression>   ::= [ <sign> ] <term> [ <adding_operator> <term> ]*
<sign>                ::= '+' | '-'
<adding_operator>     ::= '+' | '-' | '&'
<term>                ::= <factor> [ <multiplying_operator> <factor> ]*
<multiplying_operator> ::= '*' | '/' | 'mod' | 'rem'
<factor>              ::= <primary> [ '**' <primary> ] |
                          'abs' <primary> | 'not' <primary>
<primary>             ::= <literal> | <identifier> | '(' <expression> ')' |
                          <function_call>
<function_call>       ::= <identifier> '(' [ <expression>
                                             [ ',' <expression> ]* ] ')'
```


Kompileerimine

- Avaldis VHDL-s (lihtsustatud)

```
…
<relation>            ::= <shift_expression> [ <relational_operation>
                                               <shift_expression> ]
<relational_operation> ::= '=' | '/=' | '<' | '<=' | '>' | '>='
<shift_expression>    ::= <simple_expression> [ <shift_operator>
                                                <simple_expression> ]
…
```

- Võrdlustehet (avaldist) analüüsiv lõik parserist (osa kompilaatorist)

```
static boolean ParseRelation(cExpression &expression, cOperand &result)
{
  char const *op=NULL; cOperand operand1,operand2;
  if (ParseShift(expression,operand1)) return TRUE;
  if      ( _NEW_MATCH (VHDL_EQ) )  op="=";
  else if ( _REP_MATCH (VHDL_NE) )  op="/=";
  else if ( _REP_MATCH (VHDL_LT) )  op="<";
  else if ( _REP_MATCH (VHDL_LE) )  op="<=";
  else if ( _REP_MATCH (VHDL_GT) )  op=">";
  else if ( _REP_MATCH (VHDL_GE) )  op=">=";
  else { ParsePutBack(); result=operand1; return FALSE; }
  if (ParseShift(expression,operand2)) return TRUE;
  if (AddExprOper(expression,result,op,BOOL_OPER,operand1,operand2)) return TRUE;
  return FALSE;
}
```

Kompileerimine

- Alamprogrammid
  - parameetrid ja naasmisaadress pinus
- Muutujad
  - lokaalsed – pinu (*.stack*)
  - globaalsed – üldmälu (*.data*)
  - dünaamilised – (muutuv) üldmälu laiendus (*.heap*)
- Kompilaatori väljundiks on nn. objekt-fail (.o)
  - eraldi tabel märgendite jaoks (lahendab linkur)
- Kompilaatori tüüp sõltub protsessorist
  - erinevad OS-d võivad kasutada sama kompilaatorit

Kompileerimine – näide

C-programm

```
#include <stdio.h>
int main (void) {
  printf ("Hello, world!\n");
  return 0;
}
```

Assembler-koodi saamine

```
> gcc -S hello.c
```

Nimede tabel objekt-failis

```
> nm hello.o
00000000 T main
         U puts
>
```

Assembler-kood

```
    .file   "hello.c"
    .section        .rodata
.LC0:
    .string "Hello, world!"
    .text
.globl main
    .type   main, @function
main:
    leal    4(%esp), %ecx
    andl    $-16, %esp
    pushl   -4(%ecx)
    pushl   %ebp
    movl    %esp, %ebp
    pushl   %ecx
    subl    $4, %esp
    movl    $.LC0, (%esp)
    call    puts
    movl    $0, %eax
    addl    $4, %esp
    popl    %ecx
    popl    %ebp
    leal    -4(%ecx), %esp
    ret
    .size   main, .-main
    .ident  "GCC: (GNU) 4.1.2 20061115 (prerelease) (SUSE Linux)"
    .section        .note.GNU-stack,"",@progbits
```


Linkimine

- Linkur seob erinevad objekt-failid, lahendab märgendite asukohad mälus ja loob programmi mälukujutise (täidetav fail - .com/.exe/a.out/…)
- OS määrab mälu kaardi
  - programm (code) aadressist 0 (ainult loetav)
  - andmed loetavas ja kirjutatavas segmendis
- Staatiline linkimine
  - kõik alamprogrammid paiknevad mälukujutises
- Dünaamiline linkimine
  - alamprogrammid laetakse vastavalt vajadusele

Linkimine

- Objekt-fail
  - märgendid lahendamata
  - koosneb kahest osast
  1. programmi kood & andmed
     - märgendi sisu 0000
  2. märgendite tabel
     - nimi & asukohad koodis
- Linkur
  - kogub kokku globaalsed märgendid
  - otsib üles objekt-failid, kus need märgendid on
  - staatilisel linkimisel tühi koht asendatakse konkreetse aadressiga
- Mälukujutis linkurile
  - lihtsustatud

```
OUTPUT_FORMAT ("elf32-arm",
               "elf32-arm", "elf32-arm")
OUTPUT_ARCH(arm)
ENTRY(__boot__)
/* Forcing static linking */
__DYNAMIC = 0;
SECTIONS
{
  /* Commands executed at exceptions */
  . = 0x0;
  .boot : { *(.boot) }
  /* Read-only sections */
  . = 0x040;
  .text : { *(.text) } =0
  /* Data segment address adjustment */
  . = ALIGN(0x020);
  .data : { *(.data) }
  .dynamic : { *(.dynamic) }
}
```


Märgendite tabel – näide

```
/* GCD */

#include <stdio.h>

#define  MX  8
const int a[MX] = { -27, 33, 245, 121, 52, 333, 125, 422 };
static const int b[MX] = { 18, 256, 45, 11, 452, 121, 625, 312 };
int iter;

static int gcd(int xi, int yi) {
  int x=abs(xi), y=abs(yi); iter=0;
  while ( x != y ) {
    if ( x < y )  y-=x;  else x-=y;
    iter++;
  }
  return x;
}

int main() {
  int i,x;
  for (i=0;i<MX;i++) {
    x=gcd(a[i],b[i]);
    printf("%4d %4d: %4d [%d]\n",a[i],b[i],x,iter);
  }
  return 0;
}
```


```
> nm gcd.o
00000000 R a
00000020 r b
00000000 t gcd
00000004 C iter
00000068 T main
         U printf
```


OS ülesanded

- Rakenduste juhtimine
  - mäluhaldus
  - rakendustele tööaja eraldamine
    - planeerimine, ajajaotus, saalimine (swap), …
- Teenused
  - kommunikatsioon
    - rakenduste vaheline, internet, …
  - failisüsteem
  - muud päringud
    - füüsiline s/v, …