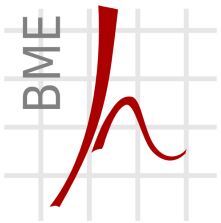
Kód optimalizálás – Utasításkészlet

Kód visszafejtés.



Híradástechnikai Tanszék

Izsó Tamás

2012. szeptember 29.

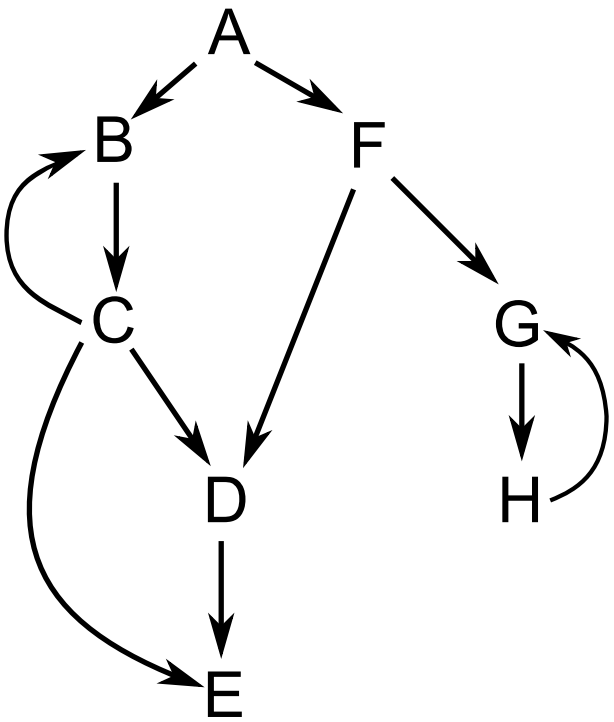
Section 1

Kód struktúra

CFG struktúrájának az analízis

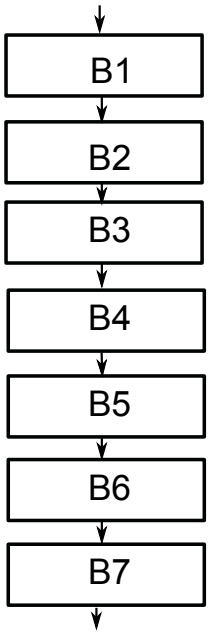
- célja felismerni a magasszintű vezérlési szerkezeteket
  - if ... then ..., if ... then ... else ..., switch ... case ...
  - for, while, repeat;
- nem struktúrált részben lévő blokkok számának a minimalizálása.

Gráf alapfogalmak

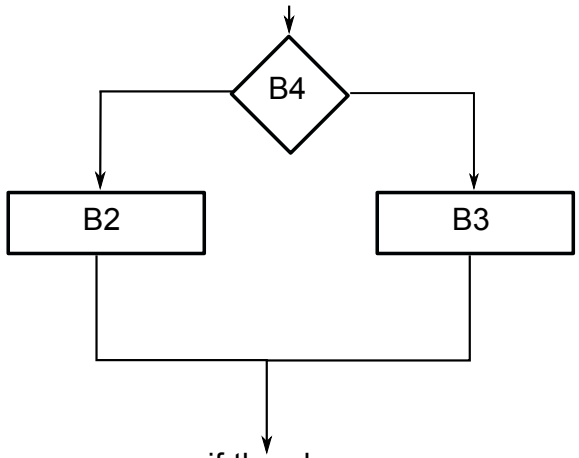


- G – irányított gráf, N – csomópontok halmaza, E – élek halmaza;
- $G = \langle N, E \rangle$;
- $N = \{A, B, C, D, E, F, G, H\}$
- $E \subseteq N \times N = \{A \rightarrow B, A \rightarrow F, B \rightarrow C, C \rightarrow B, C \rightarrow D, C \rightarrow E, D \rightarrow E, F \rightarrow D, F \rightarrow G, G \rightarrow H, G \rightarrow H\}$
- $Succ(b) = \{n \in N | \exists e \in E$ hogy $e = b \rightarrow n\}$
- $Pred(b) = \{n \in N | \exists e \in E$ hogy $e = n \rightarrow b\}$

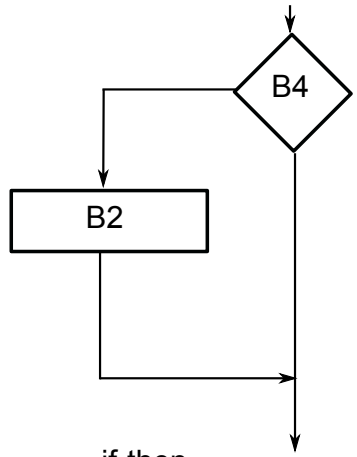
Aciklikus struktúrák



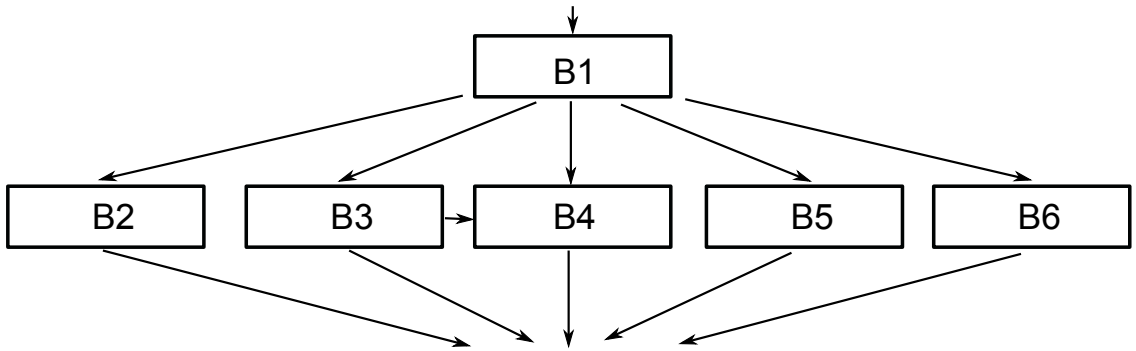
blokk



if-the-else

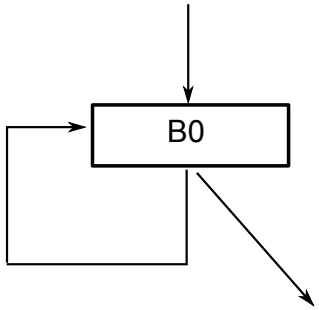


if-then

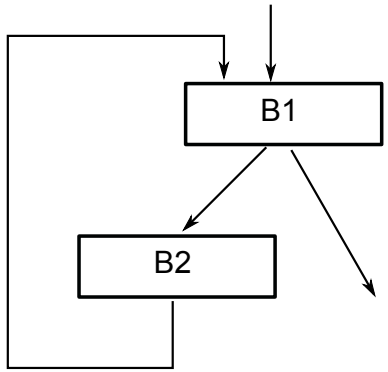


case-switch

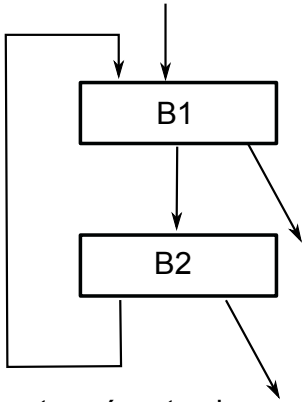
Ciklikus struktúrák



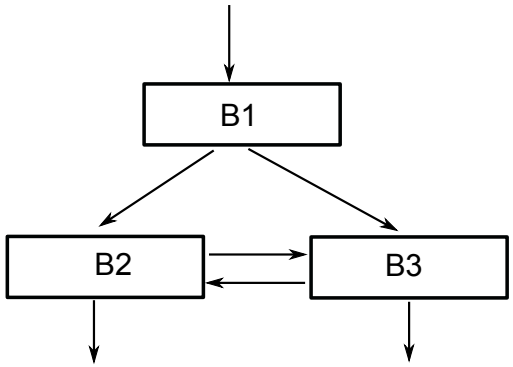
self-loop



while-loop



természetes loop



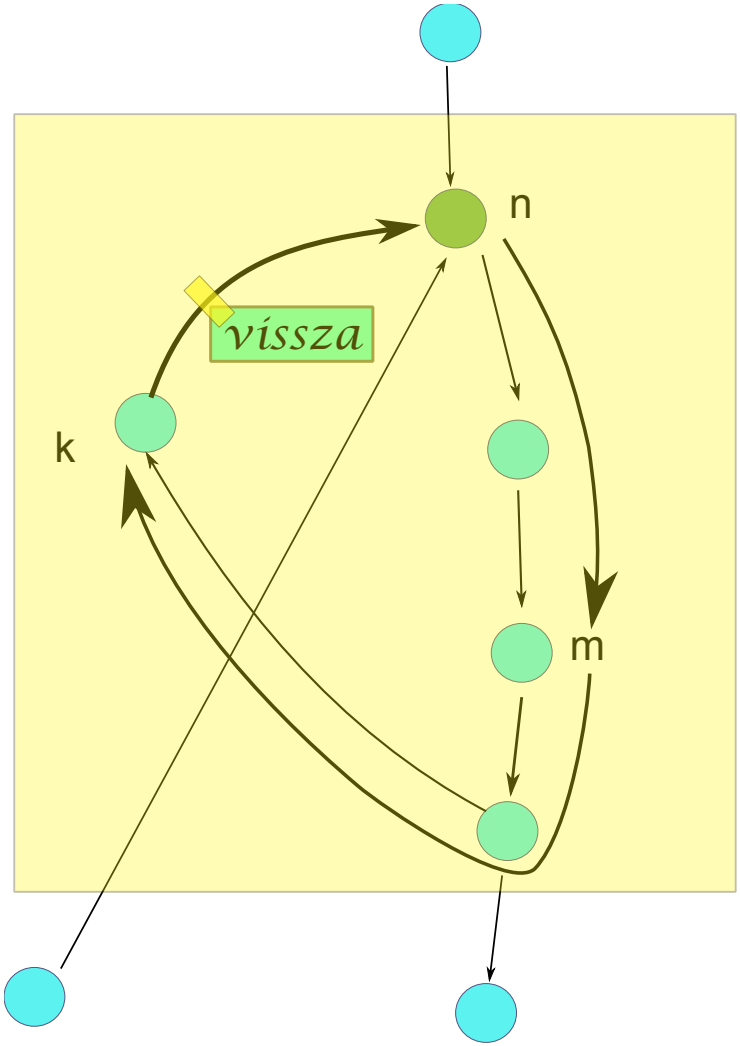
(improper) nem struktúrált szerkezet

Struktúra meghatározása

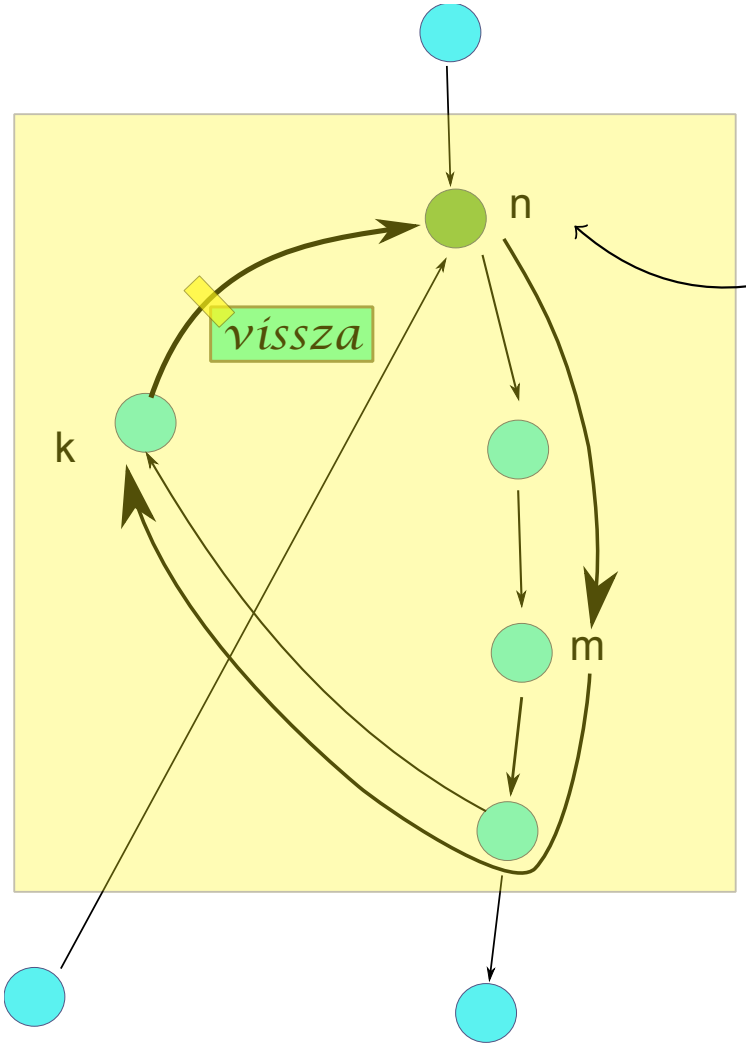
Globális adatok:

Succ, Pred : Node $\rightarrow$ set of Node;
RegionType = enum { Block, IfThen, IfThenElse, Case, Proper, SelfLoop,
WhileLoop, NaturalLoop, Improper } ;
// Blokk hozzárendelése az őt tartalmazó összevont struktúrához
StructOf: Node $\rightarrow$ Node;
// Összevont struktúra típusa (pl. if-then)
StructType: Node $\rightarrow$ RegionType;
// Összevont struktúrák halmaza
Structures: set of Node;
// Összevont struktúra alá tartozó blokkok
StructNodes : Node $\rightarrow$ set of Node;
// Összevont struktúra hierarchiája
CTNodes: set of Node;
CTEdges : set of Node x Node;
PostCtr, PostMax:integer;
Post: integer $\rightarrow$ Node;
Visit: Node $\rightarrow$ boolean;

Ciklus keresés

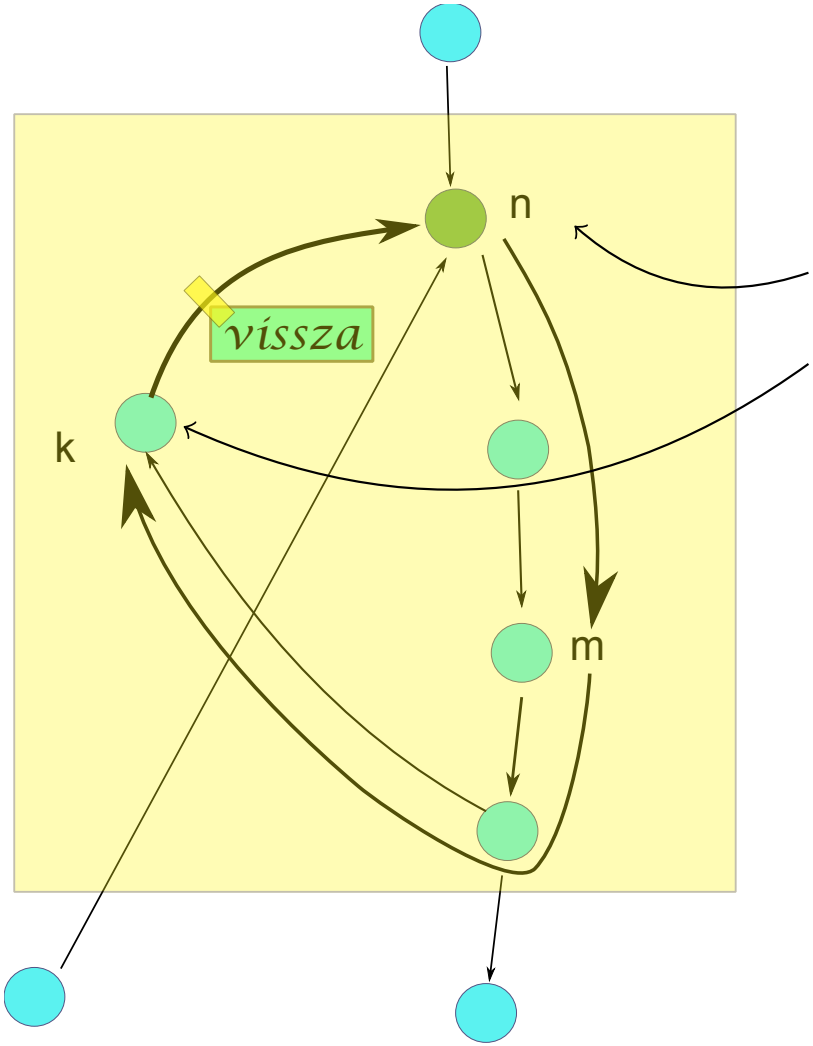


Ciklus keresés



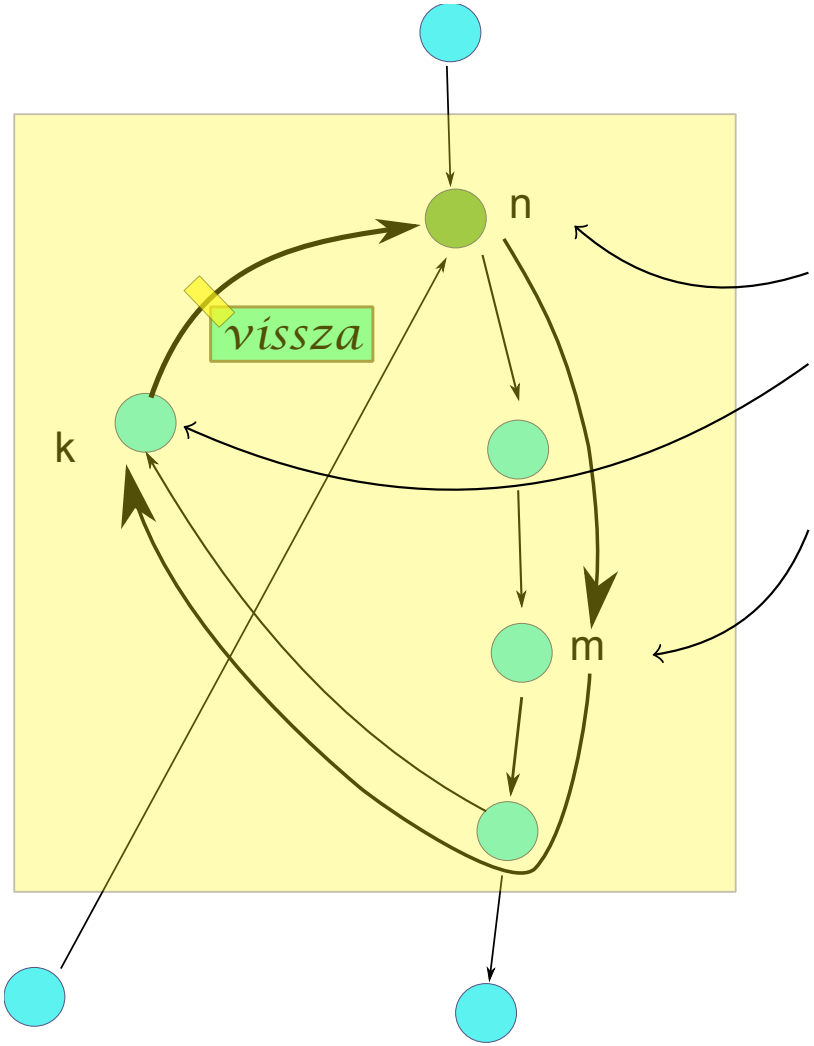
- *n* pont a ciklus kezdete;

Ciklus keresés



- n pont a ciklus kezdete;
- $\exists k \in N$ pont, ahol $Pred(n) = k$;

Ciklus keresés



- n pont a ciklus kezdete;
- $\exists k \in N$ pont, ahol $Pred(n) = k$;
- $\exists$ olyan m pont, hogy n-ből m-be és m-ből k-ba van út.

Strukturális analízis

```
Procedure StructuralAnalysis(N, E, entry)
    N : in set of Node;
    E: in set of (Node x Node);
    entry: in Node;
begin
    m, n, p : Node;
    rtype : RegionType;
    NodeSet, ReachUnder: set of Node;
    StuctOf := StructType := ∅;
    Stuctures := StructNodes := ∅;
    CTNodes := N; CTEdges := ∅;
    repeat
        Post := ∅; Visit := ∅
        PostMax := 0; PostCtr := 1
        DFS_Postorder(N, E, entry );
        while |N| > 1∧ PostCtr ≤ PostMax do
            n := Post(PostCtr) ;
            rtype :=AcyclicRType(N,E,rtype,NodeSet);
            if rtype ≠ nil then
              p:= Reduce(N,E,rtype,NodeSet) ;
              if entry ∈ NodeSet then entry := p; fi;
            else
                ReachUnder := {n};
                for each n ∈ N do
                    // ∃k amely vissza élen eljut n-be és,
                    // ∃m pont, amihez van n → m,
                    // és m → k-ba él.
                    if PathBack(m,n) then
                      ReachUnder ∪ = {m};
                    fi;
                od;
                rtype :=CyclicRType(N,E,rtype,
                    ReachUnder);
                if rtype ≠ nil then
                  p:= Reduce(N,E,rtype,ReachUnder) ;
                  if entry ∈ ReachUnder then
                    entry := p;
                  fi
                else
                  PostCtr += 1;
                fi
              fi
              od;
        until |N| = 1;
    end
```

Blokkok DFS bejárása

```
Procedure DFS_Postorder(N, E, x)
    N : in set of Node;
    E: in set of (Node x Node);
    x: in Node;
begin
    y : Node;
    Visit(x):=true;
    for each y ∈ Succ(x) do
        if !Visit(y) then
          DFS_Postorder(N,E,y);
        if;
    od;
    PostMax+= 1;
    Post(PostMax) := x;
end
```

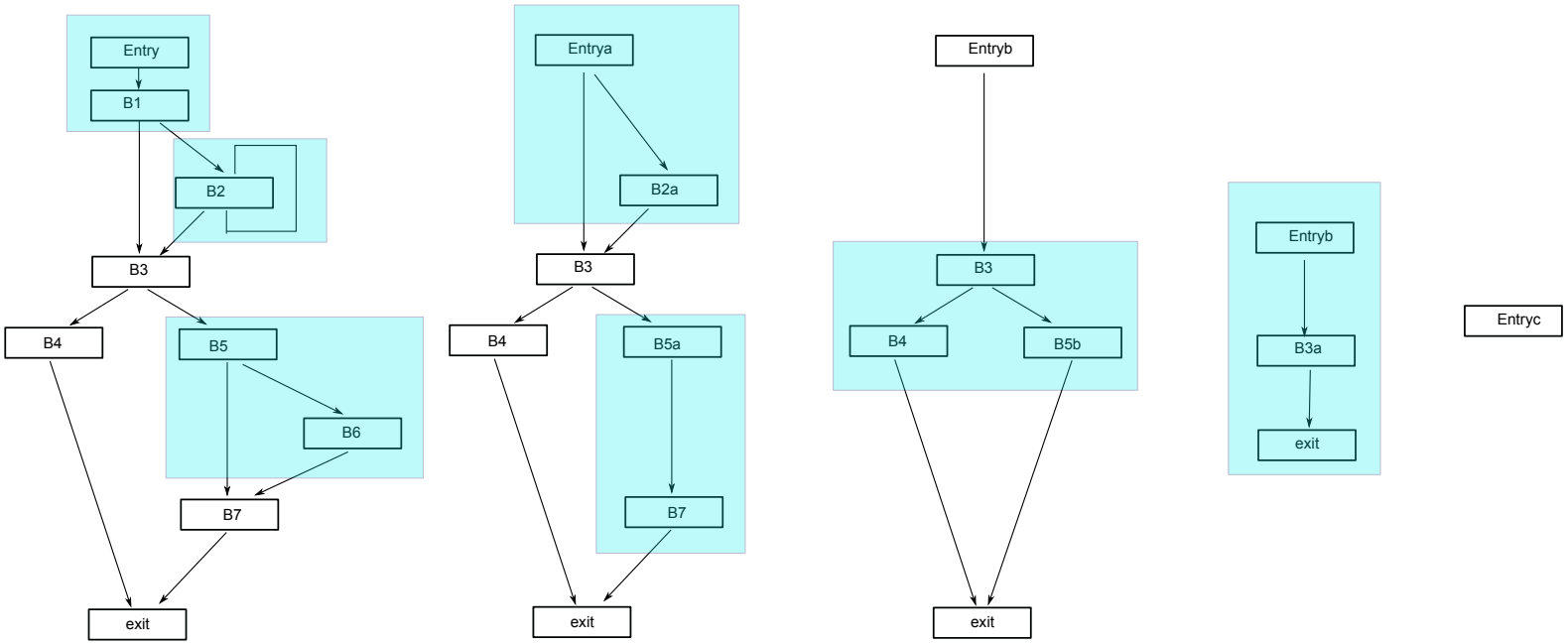
Aciklikus struktúrák feltérképezése

```
Procedure AcyclicRType(N,E,node,nset) : RegionType
    N : in set of Node;
    E: in set of (Node x Node);
    node: inout Node;
    nset: out set of Node;
begin
    m, n : Node;
    p, s: boolean;
    nset := ∅;
    n := node; p := true; s := |Succ(n)| = 1;
    while p ∧ s do
        nset ∪= {n}; n := ♦Succ(n);
        p := |Pred(n)| = 1; s := |Succ(n)| = 1;
    od;
    if p then
        nset ∪= {n};
    fi
    n := node; p := |Pred(n)| = 1; s:=true;
    while p ∧ s do
        nset ∪= {n}; n := ♦Pred(n);
        p := |Pred(n)| = 1; s := |Succ(n)| = 1;
    od
    if s then
      nset ∪= {n};
    fi
    node := n;
    if |nset| ≥ 2 then
      return Block;
    elif |Succ(node)| = 2 then
      m := Succ(node); n:= (Succ(node)-{m});
      if Succ(m) = Succ(n)
        ∧ |Succ(m)| = 1∧ |Succ(n)| = 1
        ∧ |Pred(m)| = 1∧ |Pred(n) = 1| then
        nset := {node, m, n};
        return IfThenElse;
      elif ...
        ...
      else
        return nil;
      fi;
    fi;
end
```

Ciklikus struktúrák feltérképezése

```
Procedure CyclicRType(N,E,node,nset) : RegionType
    N : in set of Node;
    E: in set of (Node x Node);
    node: in Node;
    nset: inout set of Node;
begin
    m : Node;
    if |nset| = 1 then
      if node → node ∈ E then
        return SelfLoop;
      else
        return nil;
      fi;
    fi;
    if ∃m ∈ nset !Path(node,m,N) then
      nset := MinimizeImproper(N,E,node,nset);
      return Improper;
    fi
    m := ♦(nset - { node });
    if |Succ(node)| = 2∧ |Succ(m)| = 1∧
      |Pred(node)| = 2 ∧ |Pred(m) = 1| then
      return WhileLoop;
    else
      return NaturalLoop;
    fi;
end
```

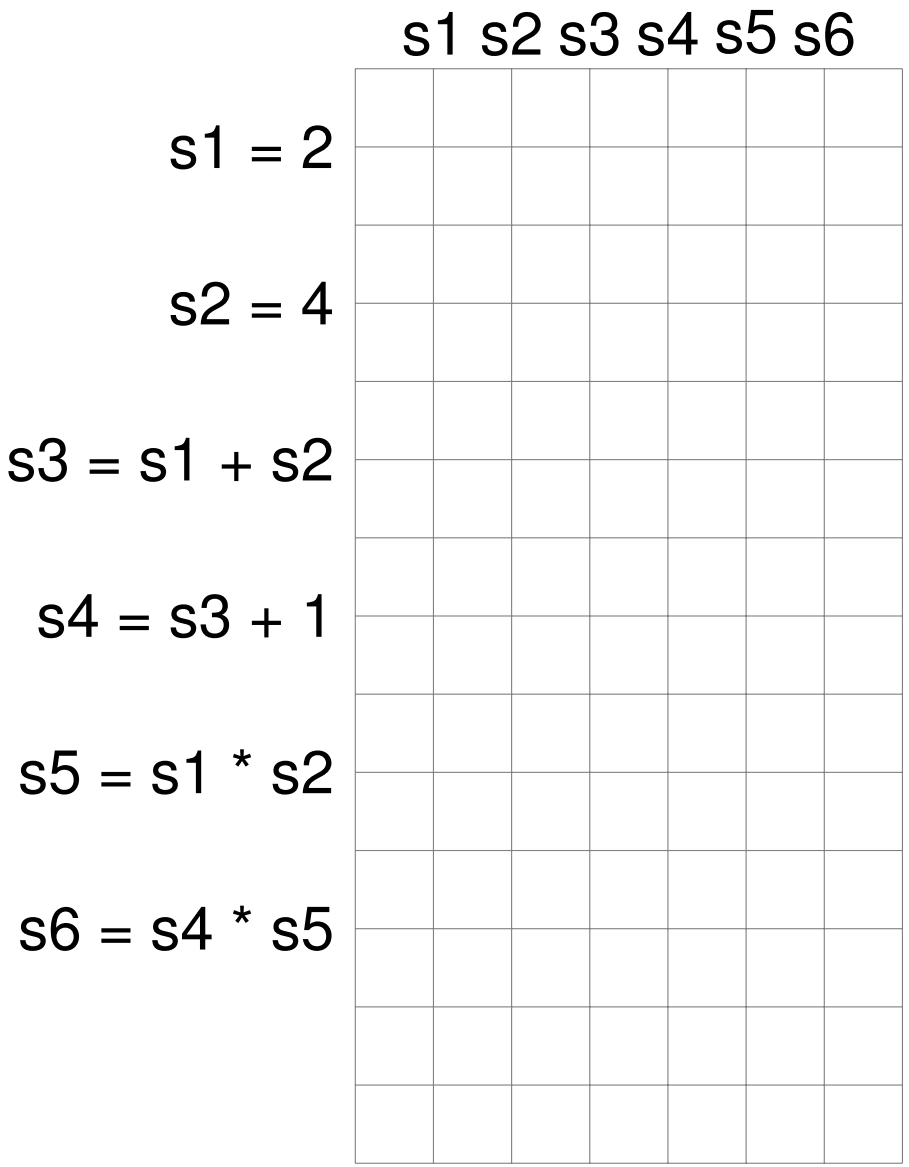
Redukció


Entry
B1
B2
B3
B4
B5
B6
B7
exit
Entrya
B2a
B3
B4
B5a
B7
exit
Entryb
B3
B4
B5b
exit
Entryb
B3a
exit
Entryc

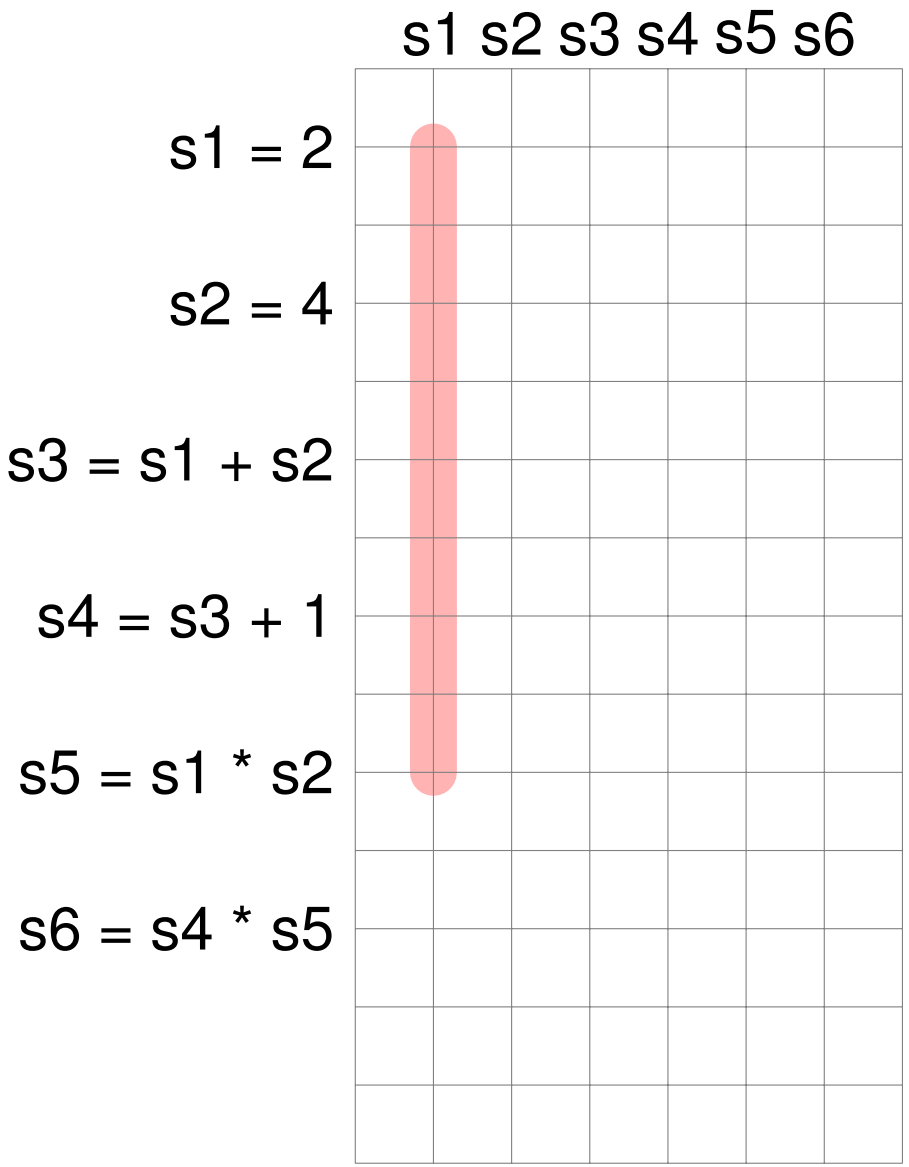
Regiszter hozzárendelés

- Intel x86 processzor kevés regisztert tartalmaz;
- néhány regiszter használata korlátozott;
- a gyors futás érdekében az adatokat érdemes a regiszterben tartani;
- ha nincs elég regiszter, ki kell menteni a memóriába;
- minimalizálni kell a egyidejűleg használt regiszterek számát;
- általában színezési problémára vezethető vissza.

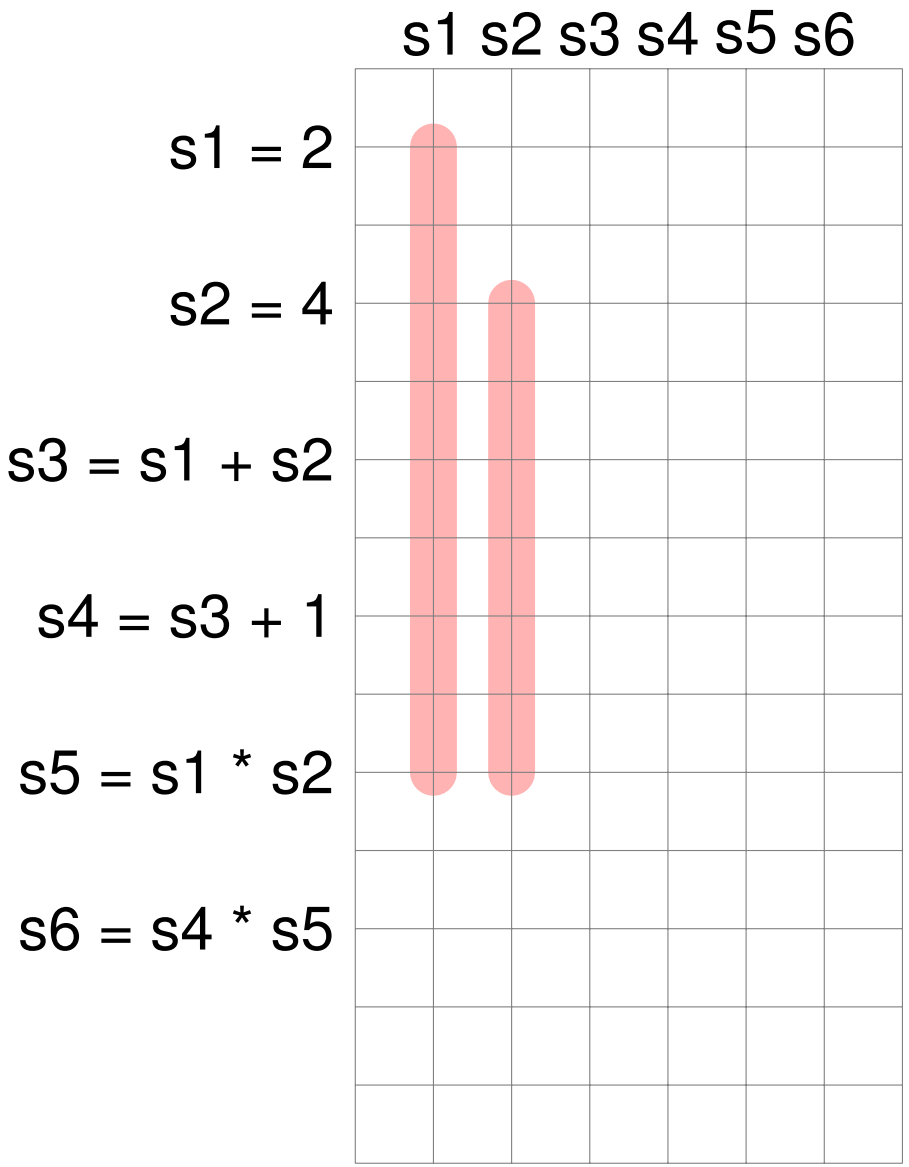
Regiszter hozzárendelés



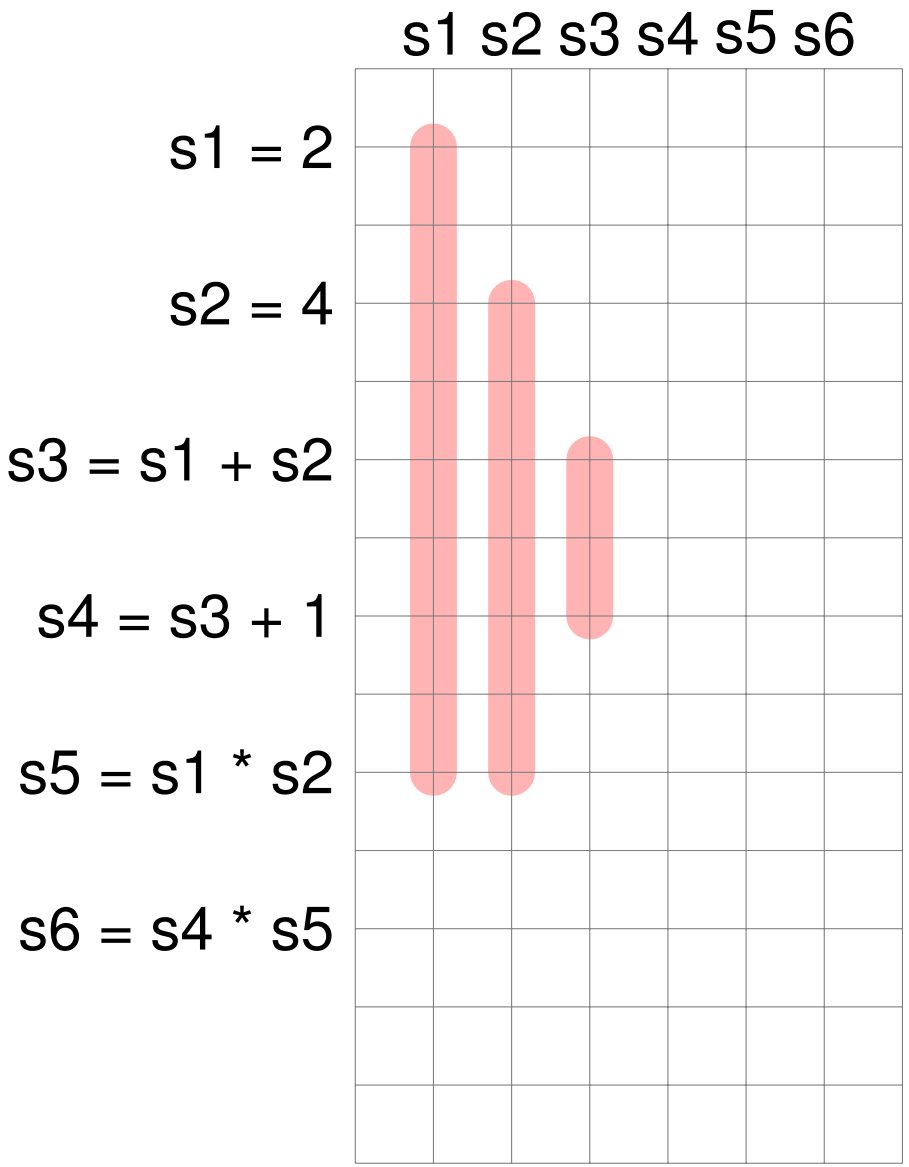
Regiszter hozzárendelés



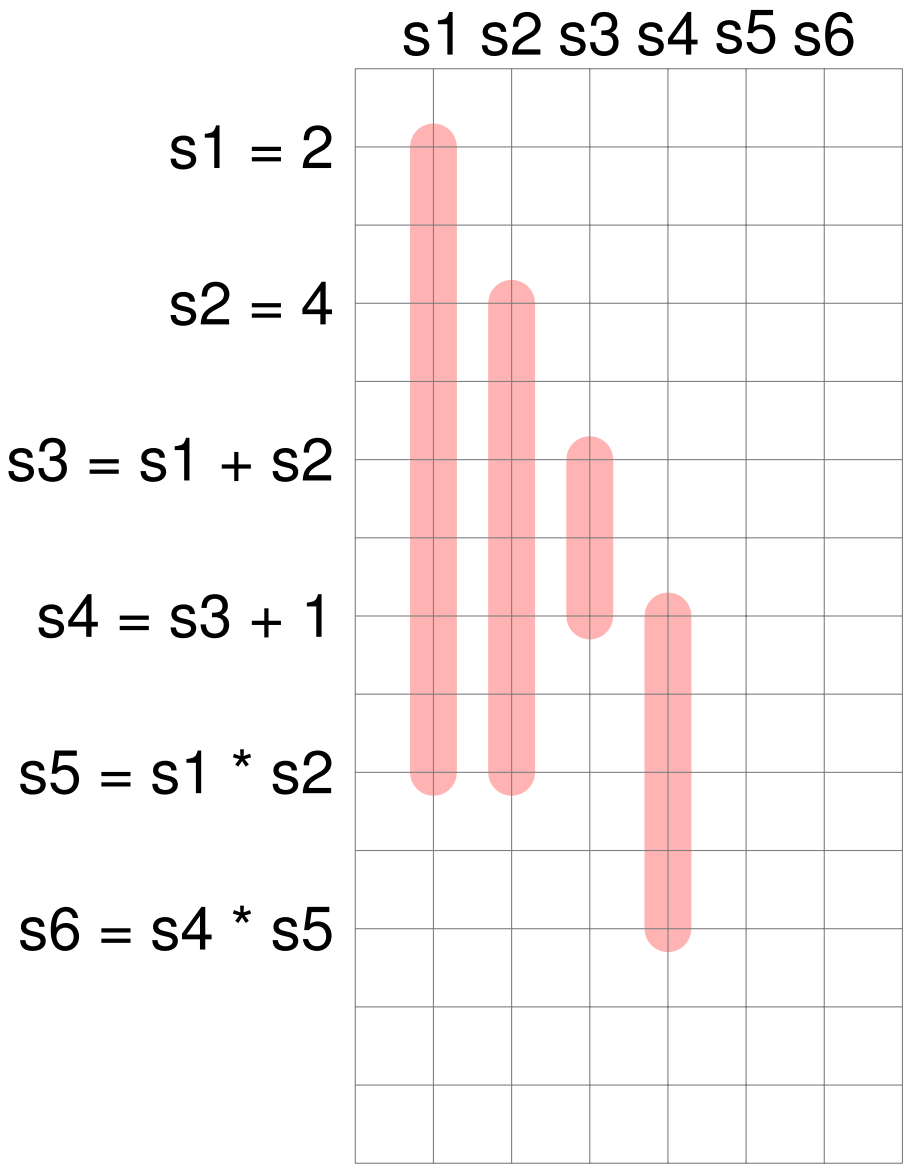
Regiszter hozzárendelés



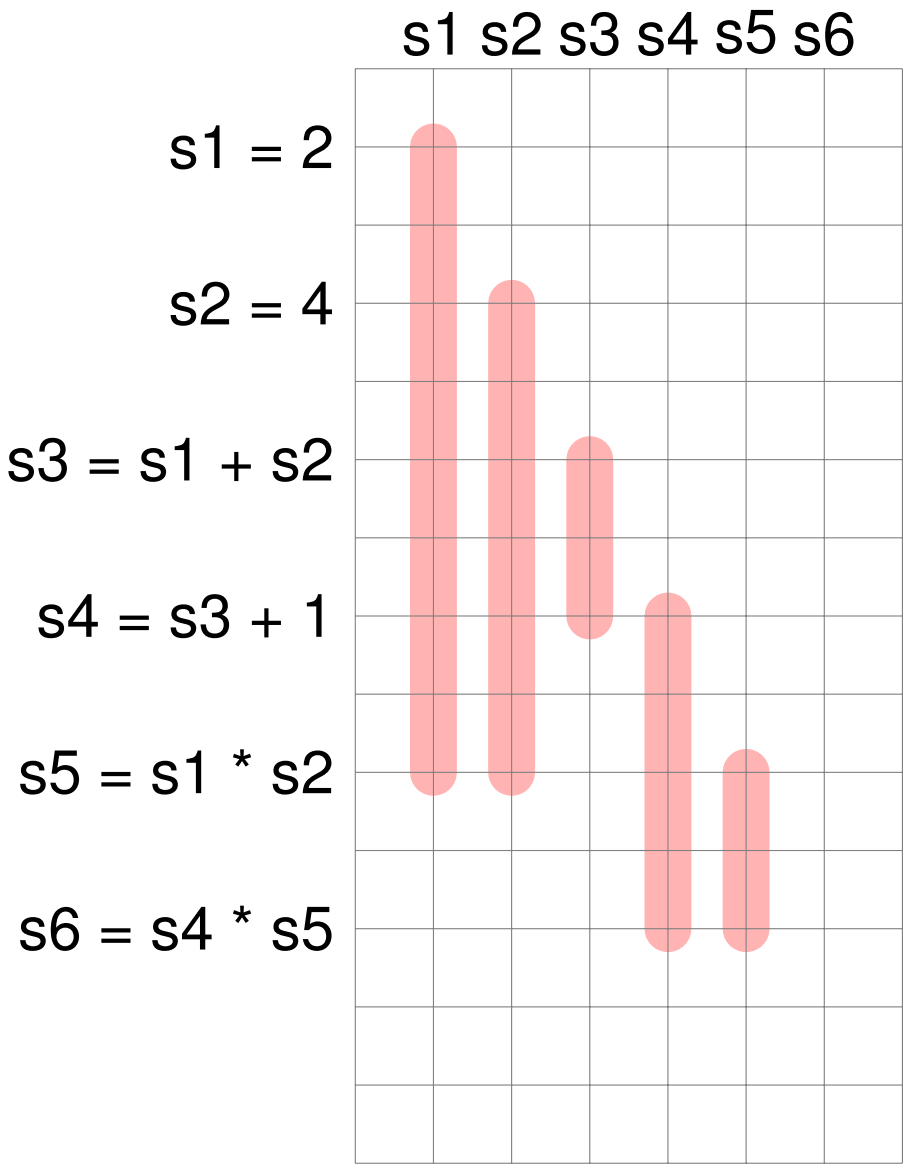
Regiszter hozzárendelés



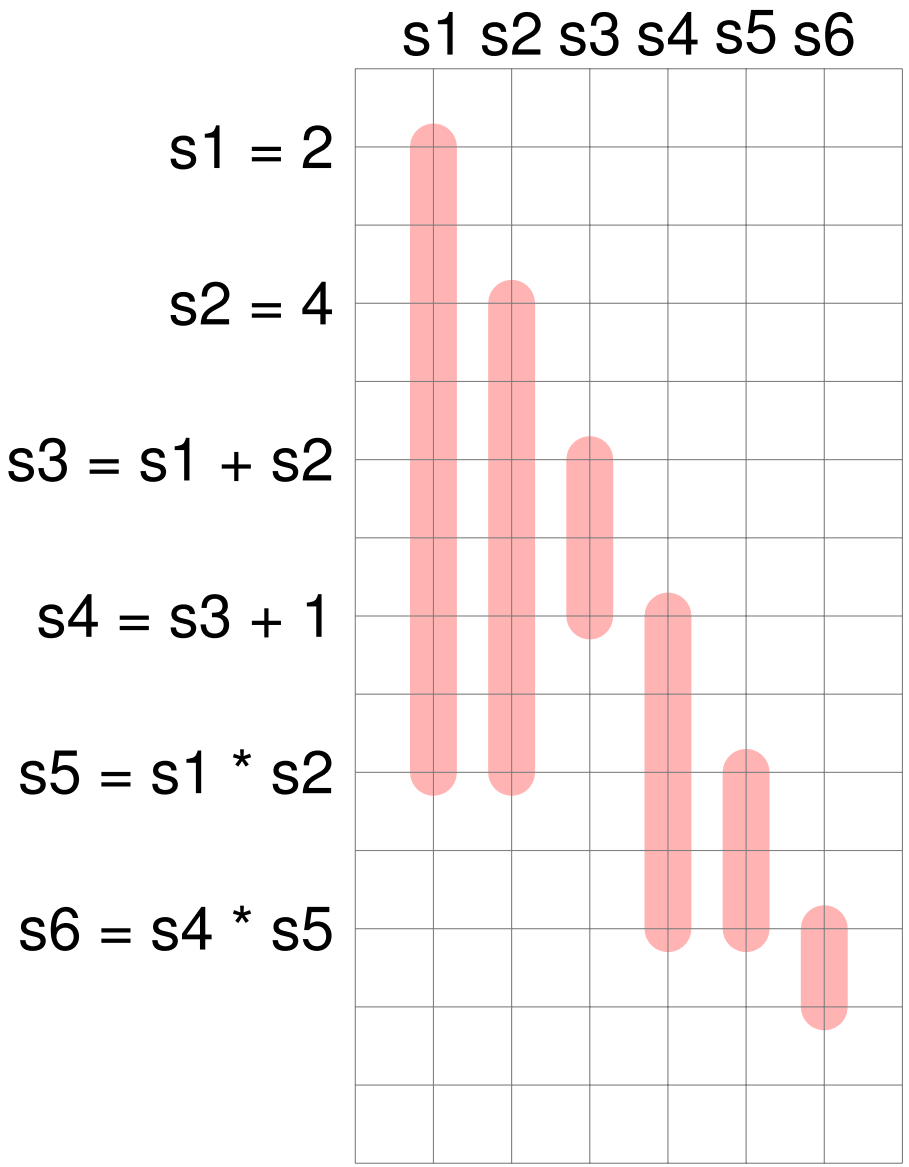
Regiszter hozzárendelés



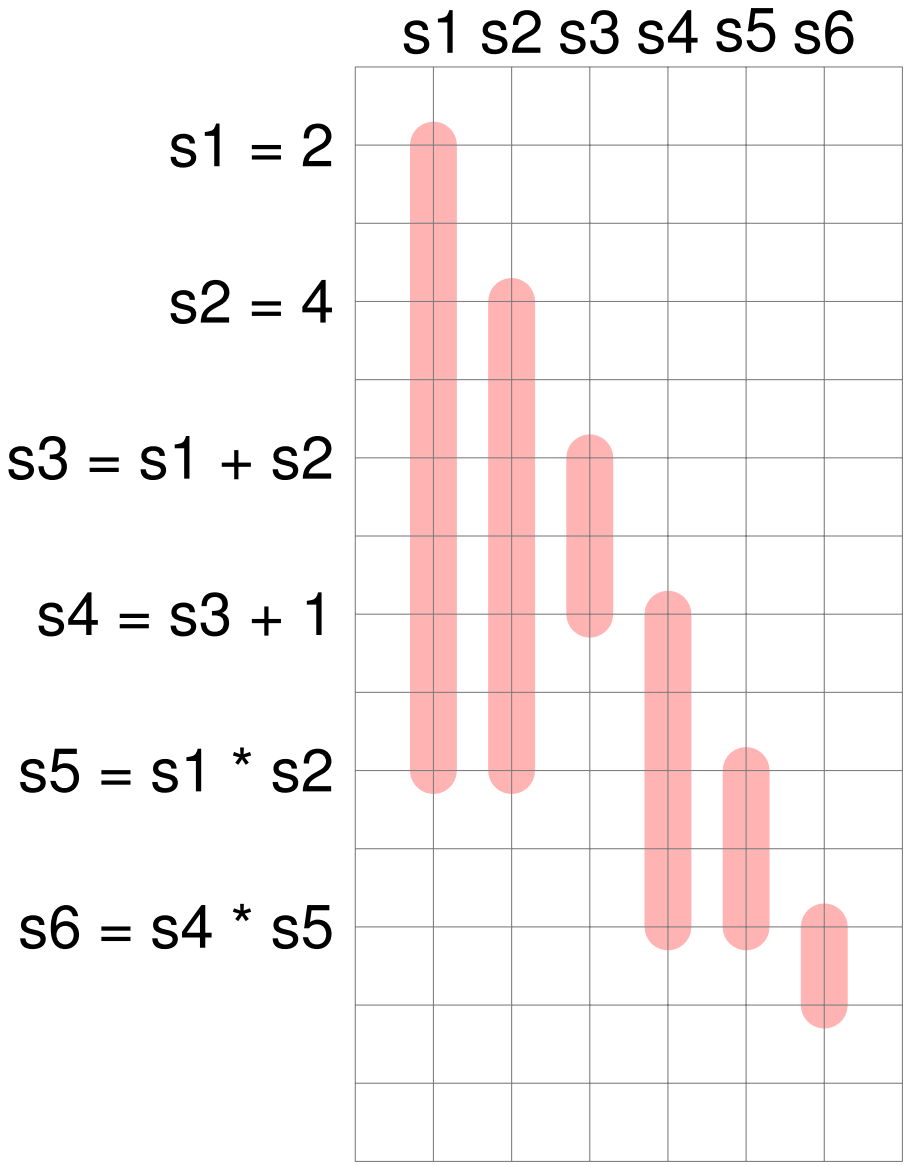
Regiszter hozzárendelés



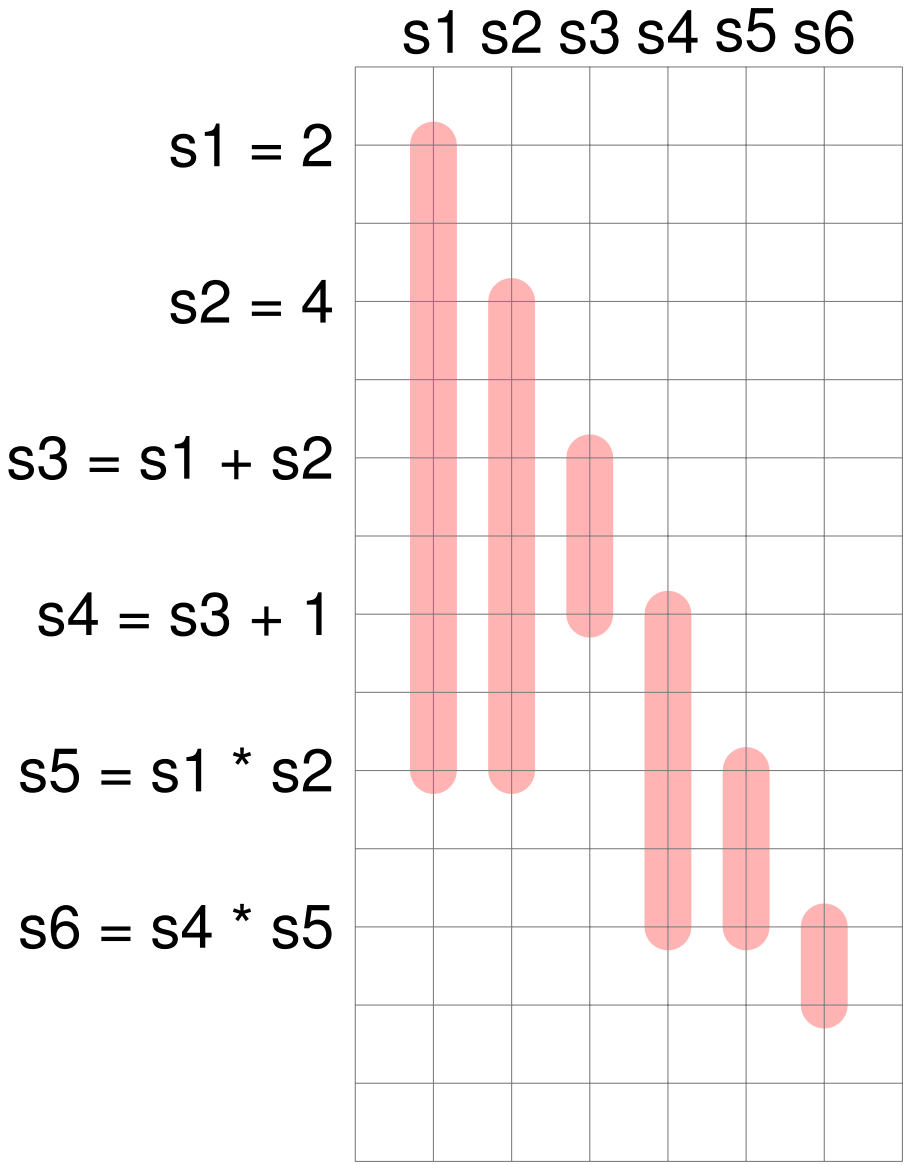
Regiszter hozzárendelés

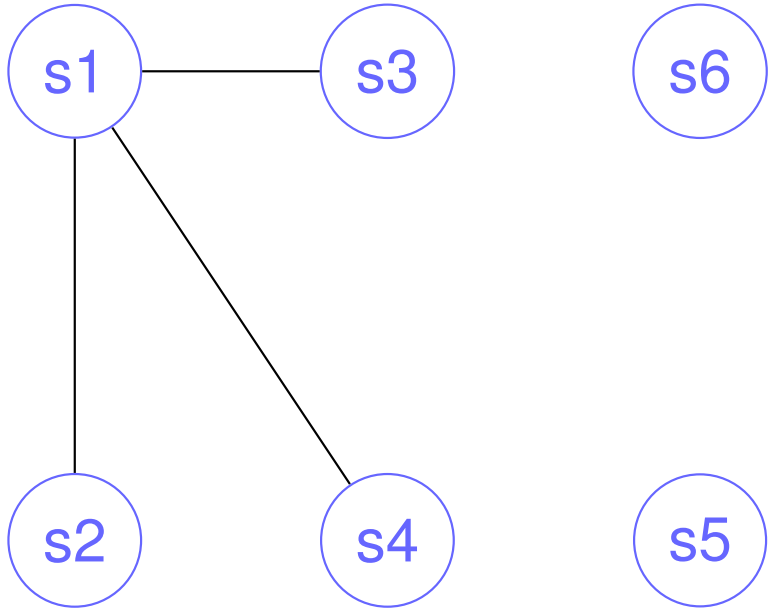


Regiszter hozzárendelés

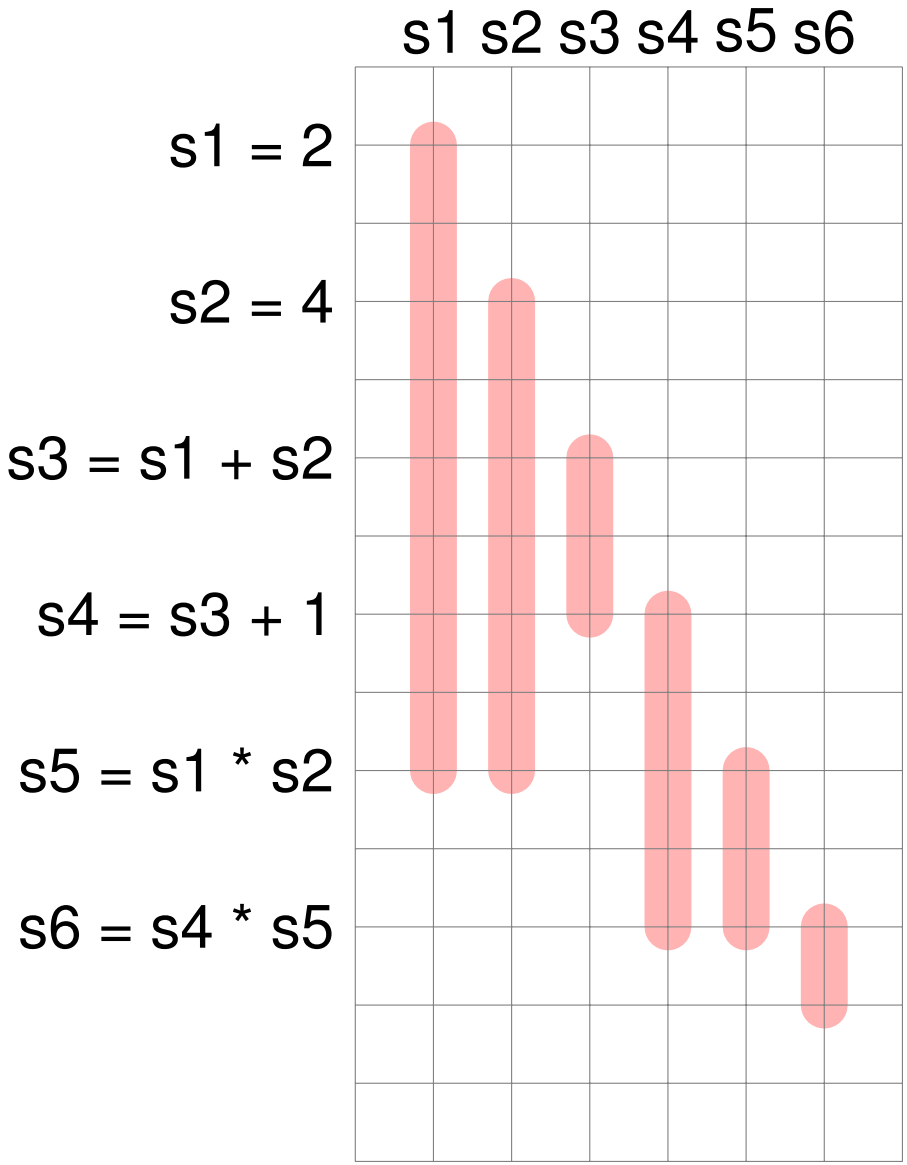


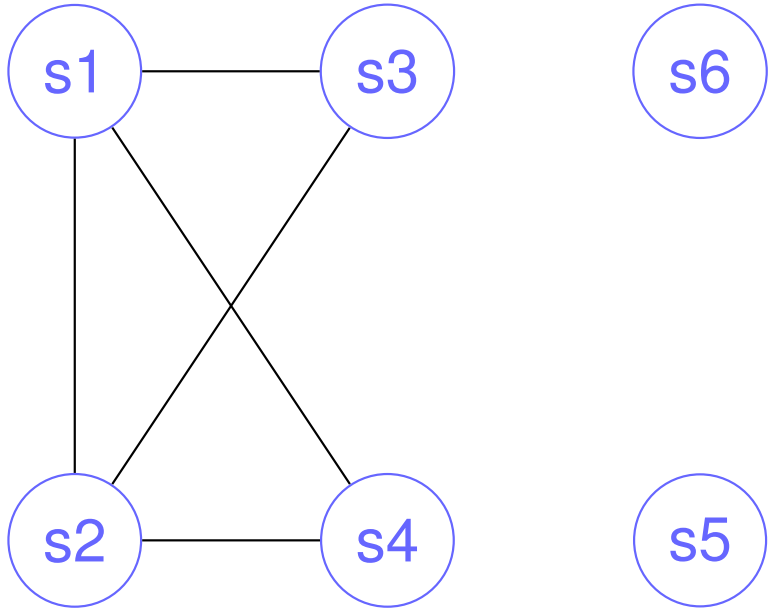
Regiszter hozzárendelés



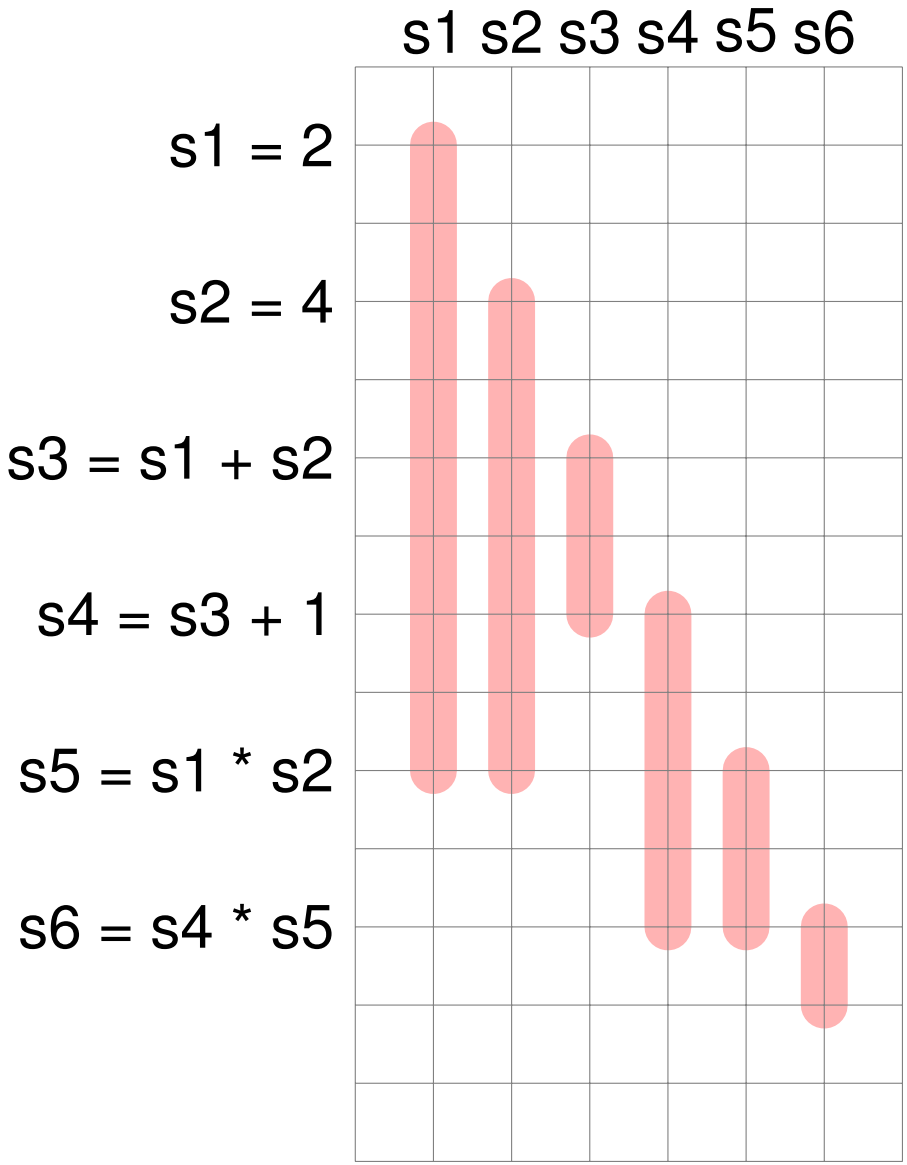


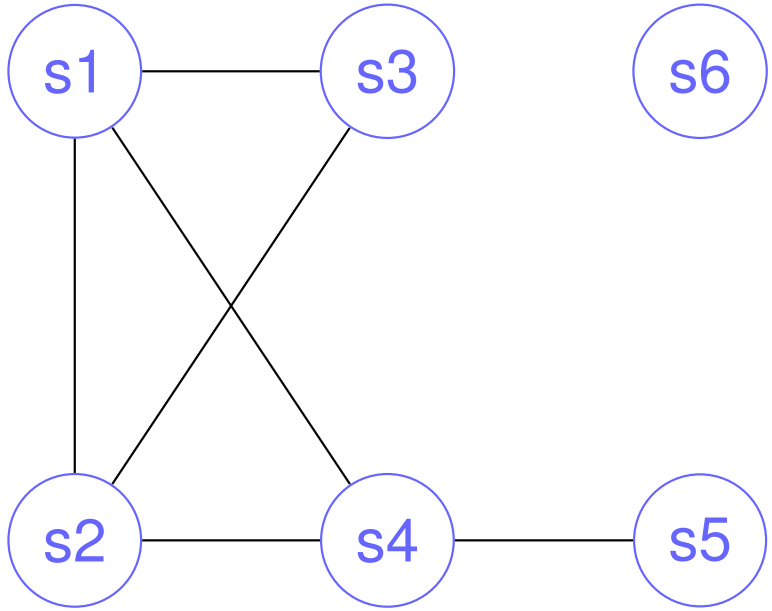
Regiszter hozzárendelés





Regiszter hozzárendelés





Egyéb optimalizálási eljárások

- aritmetikai egyszerűsítés: **x = 4 * a;** $\rightarrow$ **x = a « 2;**
- konstans kifejezés kiértékelése: **4 * 1024** $\rightarrow$ **4096**

Optimalizált kód visszafejthetősége

- megjegyzések elvesznek;
- szimbólikus nevek elvesznek;
- adattípusok elvesznek;
- ciklus átrendezés, és kifejtés (unrolling);
- különböző lokális változók azonos regiszterekben cserélik egymást ;
- algebrai átírás;
- kódmozgás;

Section 2

Intel mikroprocesszor

Intel mikroprocesszor család

- 1978-ban dobták piacra a 8086, 8088, 80186 processzorokat;
- 16-bites regiszterek;
- szegmens-offset címzés;
- szegmens regiszter állítás nélkül 64Kbyte címezhető;
- néggyel eltolt szegmensregiszter + a 16 bit offset 20 bites címet alkot;
- 8087 lebegőpontos koprocesszor.

Intel 286 mikroprocesszor

- 1982-ben vezették be;
- protected mód;
- laptábla mérete 24-bit;
- 16Mbyte memória címezhető.

Intel 386 mikroprocesszor

- 1985-ben vezették be;
- 32 bites regiszterek;
- 32 bites cím, 4 GByte címezhető;
- lapozást támogatja;
- szegmens kezelés kikerülhető, flat memóriamodell;
- linux futtatható rajta.

Intel 486 mikroprocesszor

- 1989-ben vezették be;
- DX verzióba beintegrálták a lebegőpontos koprocesszort;
- gyorsítótár (cache) alkalmazása;
- pipeline utasítás feldolgozás;

Intel Pentium mikroprocesszor

- 1993-ben vezették be;
- gyorsítótár méretét megduplázták, felét a kód felét az adatok számára használták;
- két pipeline utasításfeldolgozó egység;
- feltételes ugrásokra elágazásbecslést alkalmaztak
- két processzoros mód támogatása
- MMX utasításkészlet

Intel Pentium mikroprocesszor

- 1995-1999 között jelent meg a Pentium 6, gyártástechnológia fejlődik, gyorsabb az elődöknél;
- 2000-2006 NetBurst mikroarchitectura, SSE3 utasítások;
- 2003 Pentium-M kis fogyasztás;
- 2004 64 bites processzor 40 bites fizikai cím 1 Tbyte címezhető, 8-ról 16-ra növelték az általános célú regiszterek számát;
- 2005-2007 két mag, 64 bit ;
- stb.

Section 3

Utasításkészlet

Utasításkészlet tervezésének szempontjai

- technológiából adódó kötöttségek (tranzisztorok száma);
- chipek költsége;
- fogyasztás;
- utasításkészlet bővíthetősége;
- előző sorozat kompatibilitásának a felvállalása;
- új utasítások és működési elvek oktatása.

RISC vs CISC

- CISC Complex Instruction Set Computing
  - utasítások hossza változik;
  - műveletek végzés regiszterek és memória között;
  - kevés számú regiszter.
- RISC (Reduced Instruction Set Computing)
  - utasítások hossza azonos;
  - műveletek végzés csak regiszterek között;
  - sok regiszter.

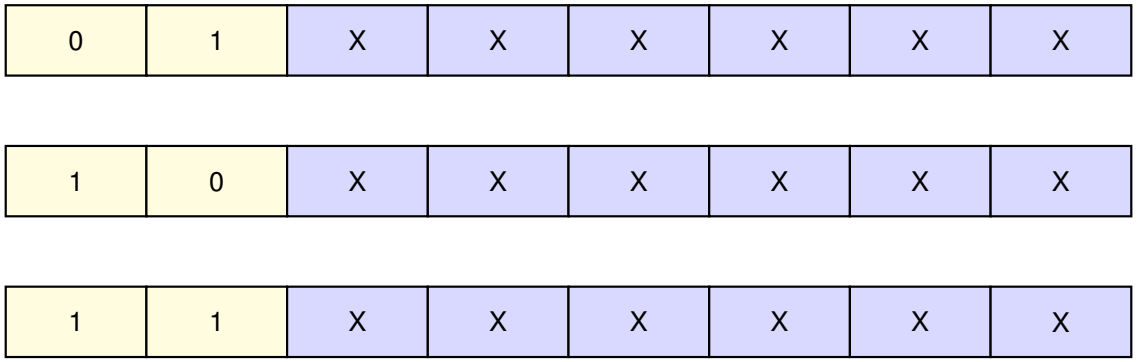
Intel 80x86 processzor

- Utasításkészlete CISC tulajdonságot mutat.
- Belül a CISC utasításokat egyszerű mikróműveletekre bontja (μ Op) és párhuzamosan képes ezeket végrehajtani. (pl. Sandy Bridge Pipeline architektúra).

Gépi utasítás mérete

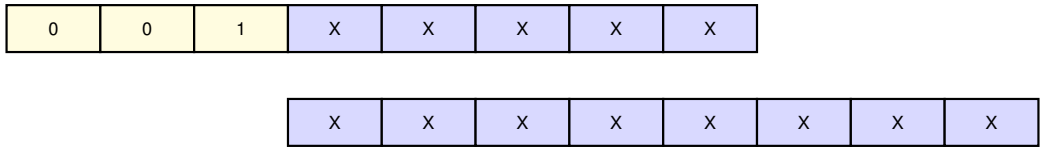
- 1 byte, 256 utasítás, kevés
- 2 byte, 65536 utasítás, sok
- 1.5 byte az utasítások átlagos hossza

Változó hosszúságú utasításkészlet *lehetséges* megvalósítása:

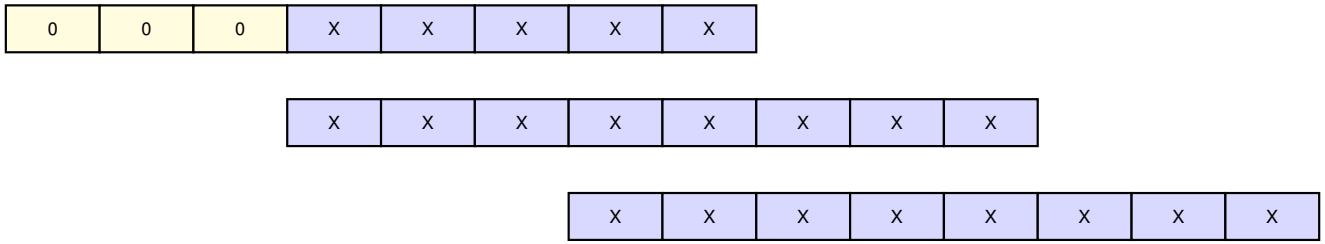


Egy byte-os utasítások száma 3*x*64

Gépi utasítás mérete



2 byte-os utasításoknál 16 bitből 3-at lerögzítünk, így 2^{13} azaz 8192 db lehetséges.



3 byte-os utasítások maximális száma 2^{24-3}.

Általános utasítások csoportosítása

- adatmozgató utasítások (**mov**)
- fixpontos aritmetikai utasítások (**add**, **sub**)
- decimális utasítások (**daa**, (**das**))
- logikai utasítások (**and**, **or**, **not**)
- hasonlító utasítások (**cmp**)
- léptető utasítások (**shl**, **rol**)
- bit és byte kezelő utasítások (**bt**, **sets**)
- feltételes és feltétel nélküli vezérlésátadó utasítások (**jmp**, **jne**, **call ret**, **int**)
- stringkezelő utasítások (**movs**, **scas**)
- input és output utasítások (**in**, **out**)
- flag beállító utasítások (**stc**, **clc**)
- (**enter**, **leave**) utasítások
- szegmesregiszter utasítások (**lds**, **les**)
- egyéb (**lea**, **nop**)

Lebegőpontos utasítások

- FPU adatmozgató utasítások (**fld**, **fst**)
- lebegőpontos aritmetikai utasítások (**fadd**, **fsub**)
- hasonlító utasítások (**fcom**)
- függvények (**fsin**)
- FPU-t vezérlő utasítások (**finit** , **fnop**)

Utasítások hozzárendelése a műveletkódhoz

- utasításcsoport megkülönböztetése 4 bit;
- csoporton belüli megkülönböztetés 3 bit;
- operandusok megadása regiszter esetén $2x3$ bit;
- gyakran használt utasítások (**mov**(**EAX**,displ) legyenek 8 bitesek

Így már bőven túlléptük a 8 bitet!