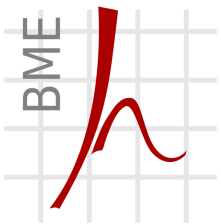
Vezérlésfolyam gráf és X86 utasításkészlet

Kód visszafejtés.



Híradástechnikai Tanszék

Izsó Tamás

2016. november 3.

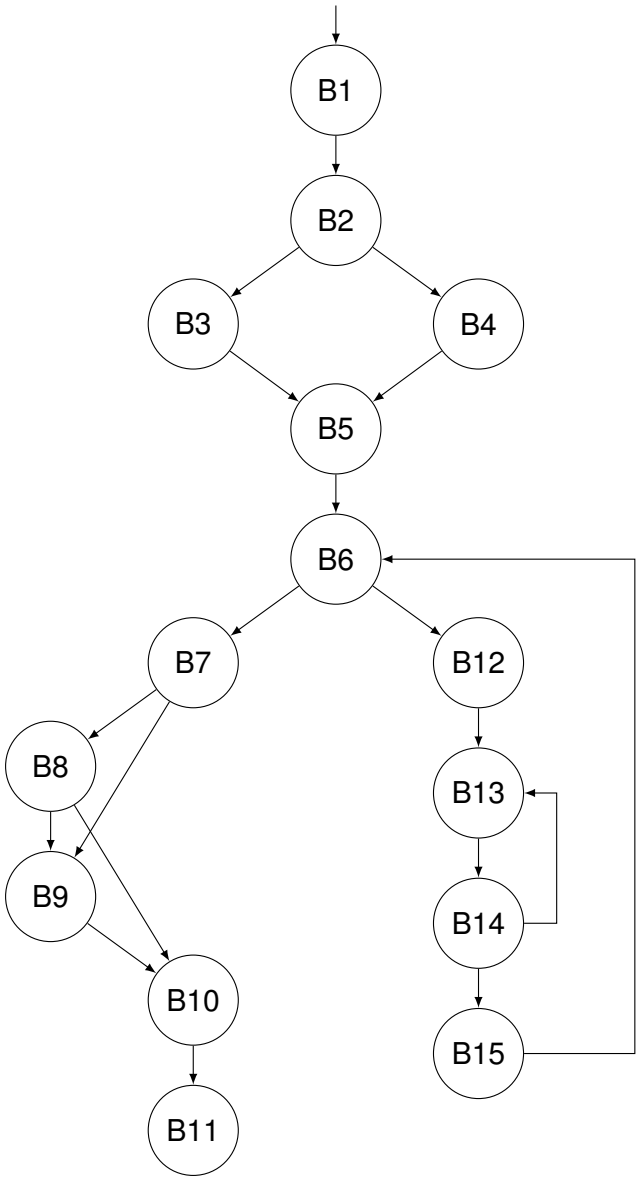
Intervallum algoritmus

Procedure Intervals(G={N, E, h})
    $\Im := \{\}$
    $H := \{h\}$
    for($\forall n$ nem bejárt pontra $n \in H$) do
        $I(n) := \{n\}$
        repeat
            $I(n) := I(n) + \{m \in N$ ahol $\forall p \in Pred(m)$ pontra teljesül $p \in I(n)\}$
        until nincs több csomópont, ami hozzáadható $I(n)$ *hez*
        $H := H + \{m \in N \mid m \notin H \ \bigwedge \ m \notin I(n) \ \bigwedge \ (\exists p \in Pred(m)$ amelyre teljesül hogy $p \in I(n))\}$
        $\Im = \Im + I(n)$
    end for
end procedure

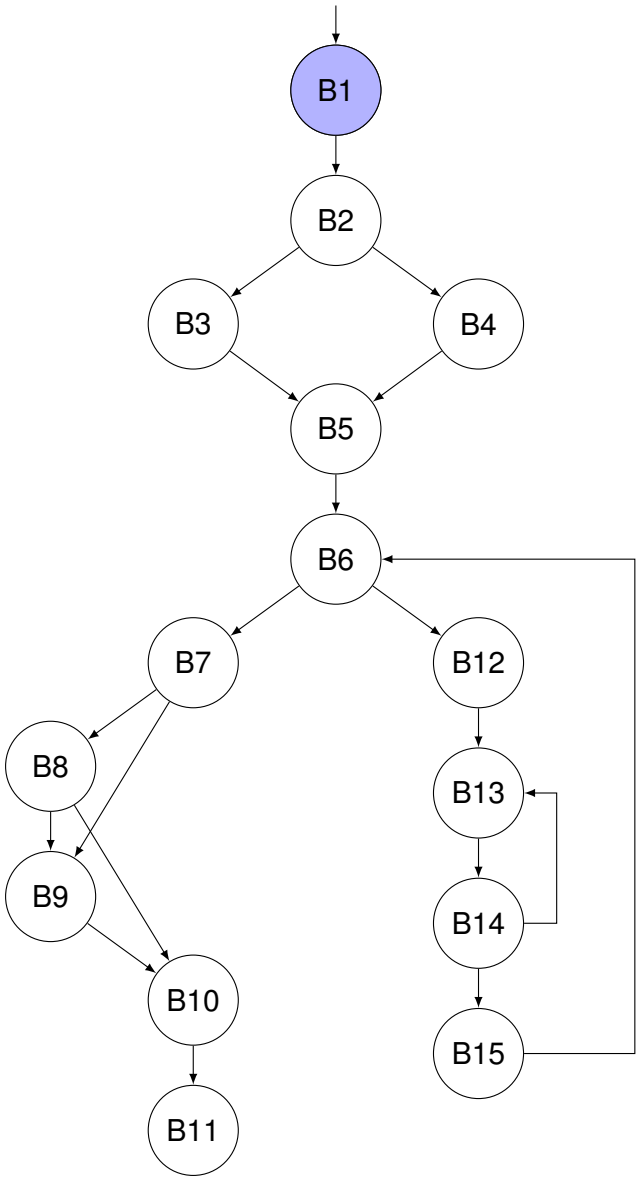
Megjegyzés az intervallum algoritmushoz

- Az eljárás egy vezérlésfolyam gráfot $G = \{N, E\}$ és egy kezdőpontot h kap, és az intervallumok halmazát $\Im$ adja vissza.
- Azokat a csomópontokat tesszük egy intervallumba, akikbe olyan helyről mutat él, amelyeket már felvettünk az aktuális intervallumba (belső repeat ciklus).
- Azok a pontok, amelyekre az intervallumban lévő csomópontokból mutat él, egy következő intervallum kezdőpontját képzik, és ezek a H (head) halmazba kerülnek.

Intervallum analízis szemléltetése

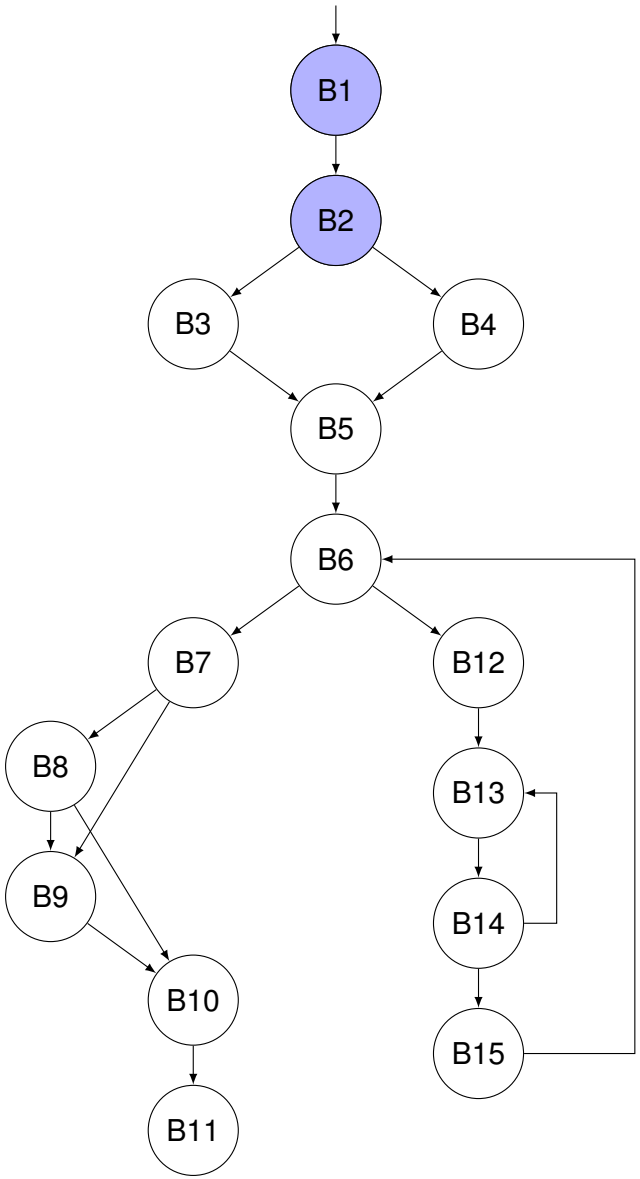


Intervallum analízis szemléltetése



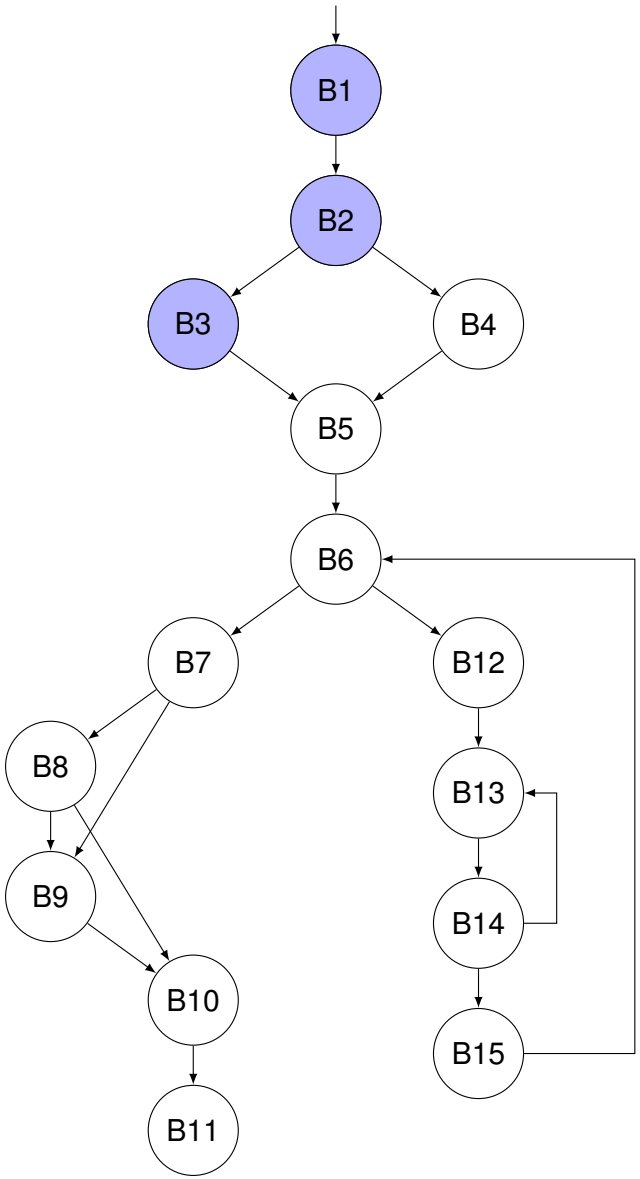
$H_1 = \{B1\}$ $I_1 = \{B1\}$

Intervallum analízis szemléltetése



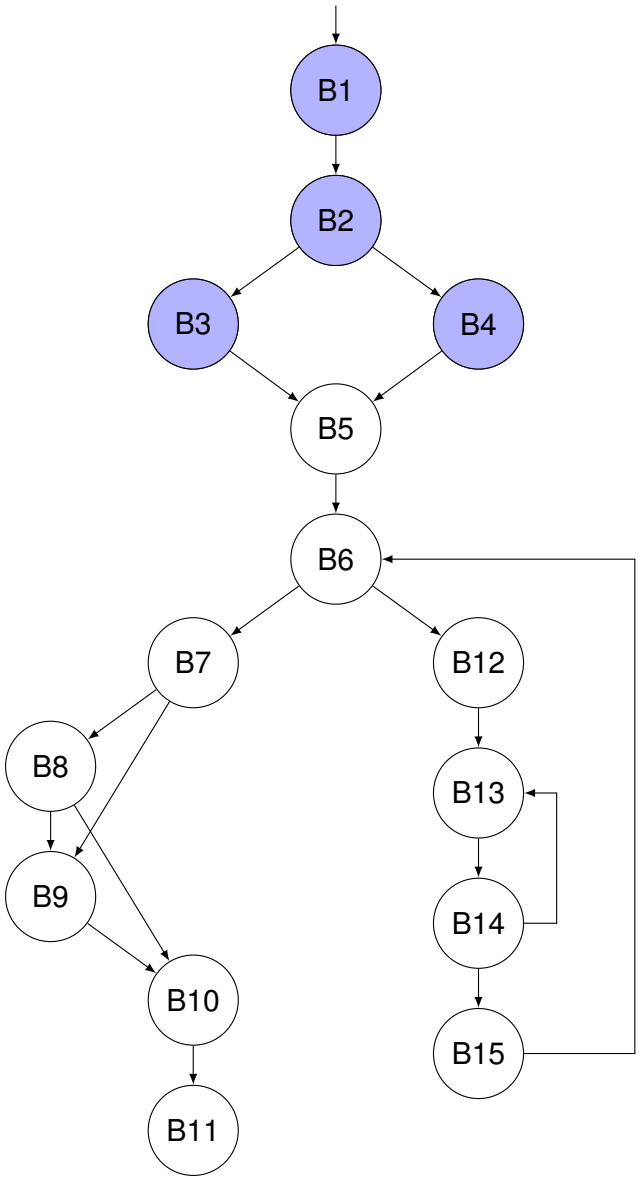
$H_1 = \{B1\}$ $I_1 = \{B1, B2\}$

Intervallum analízis szemléltetése



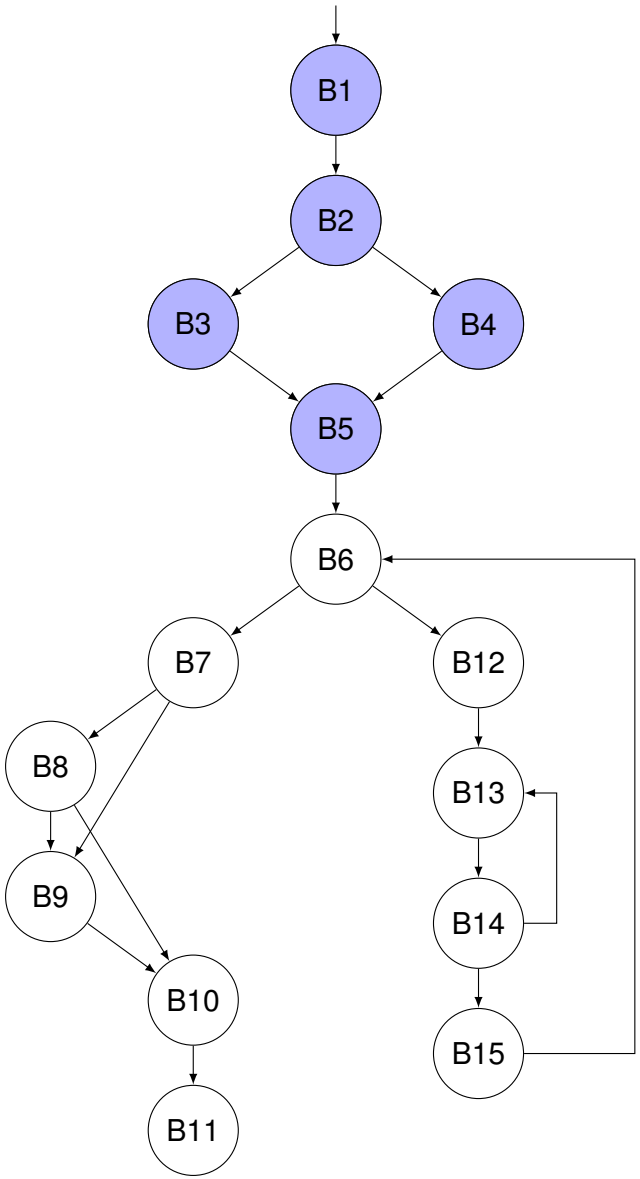
$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3\}$

Intervallum analízis szemléltetése



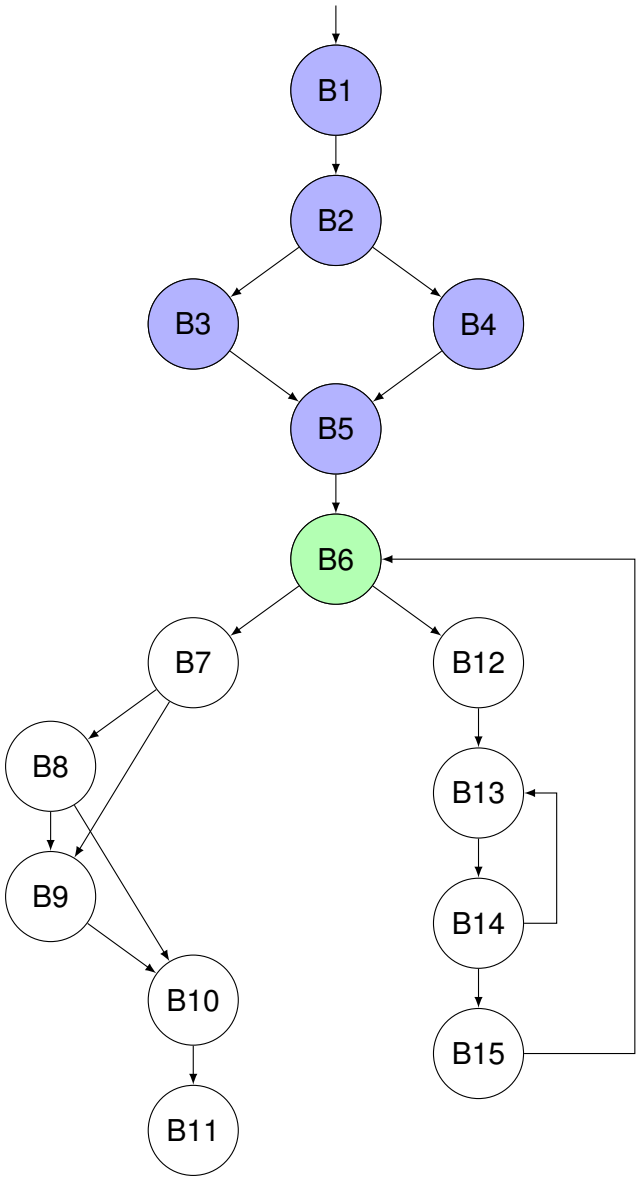
$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4\}$

Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

Intervallum analízis szemléltetése

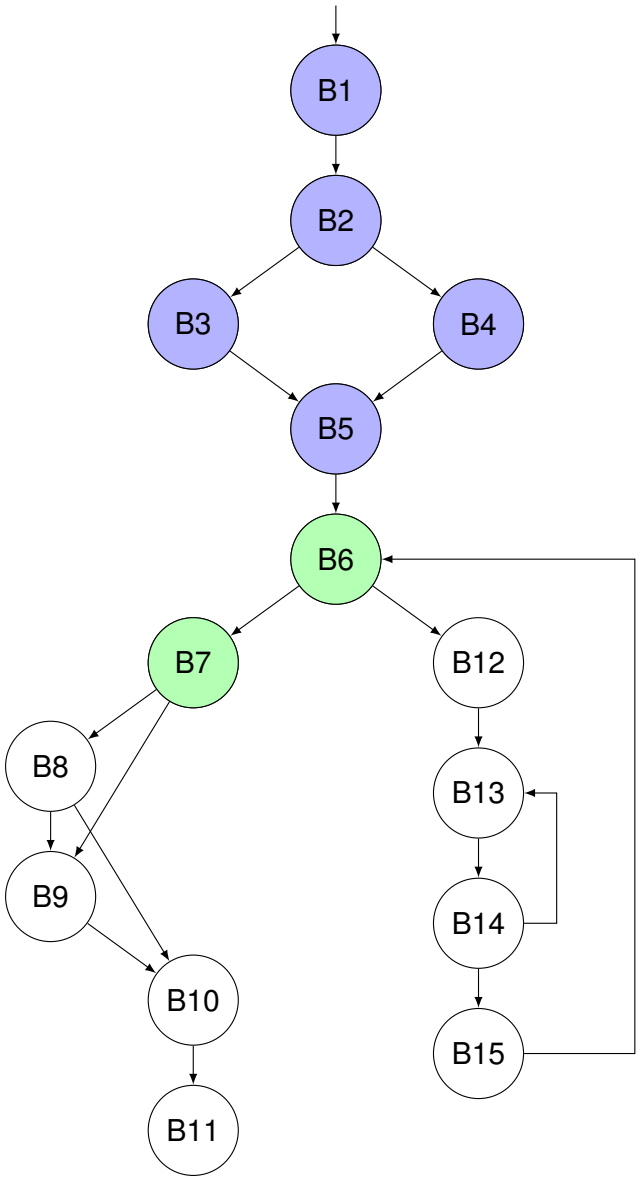


$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6\}$

$B15$ pont nincs az I_1 halmazban
ezért új intervallumot kell létrehozni!

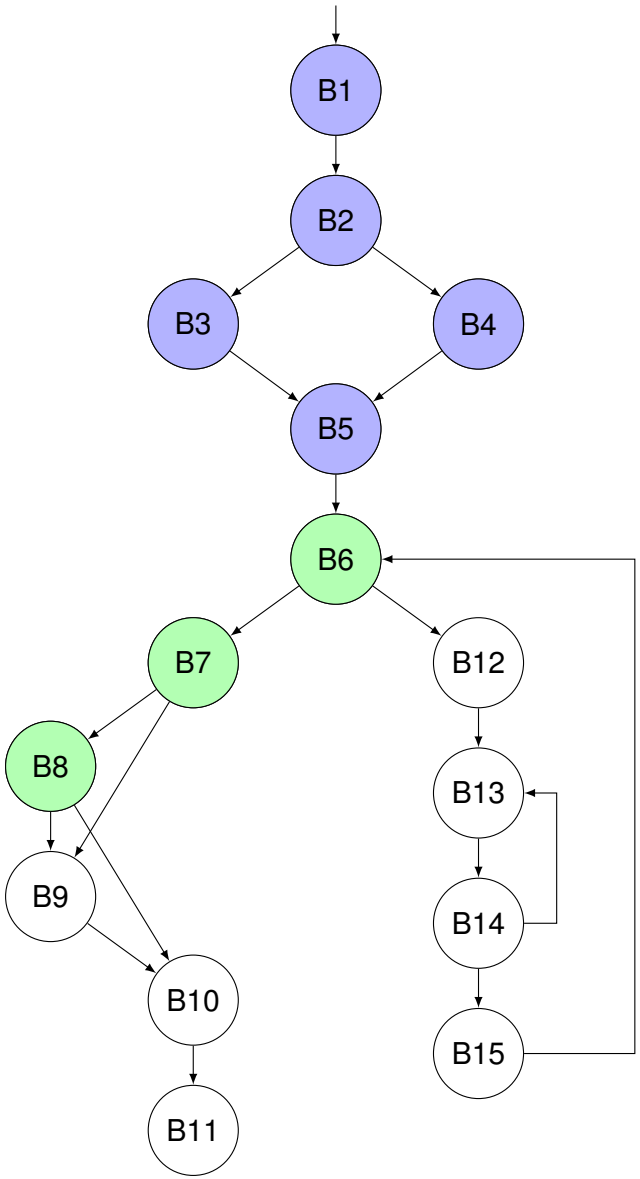
Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7\}$

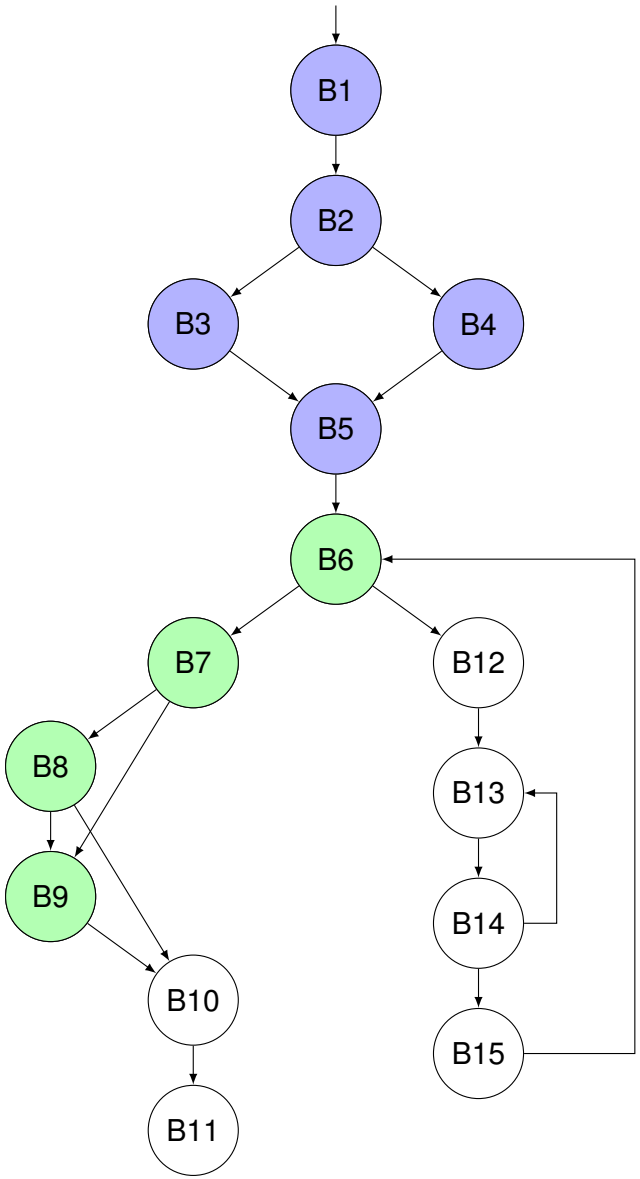
Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8\}$

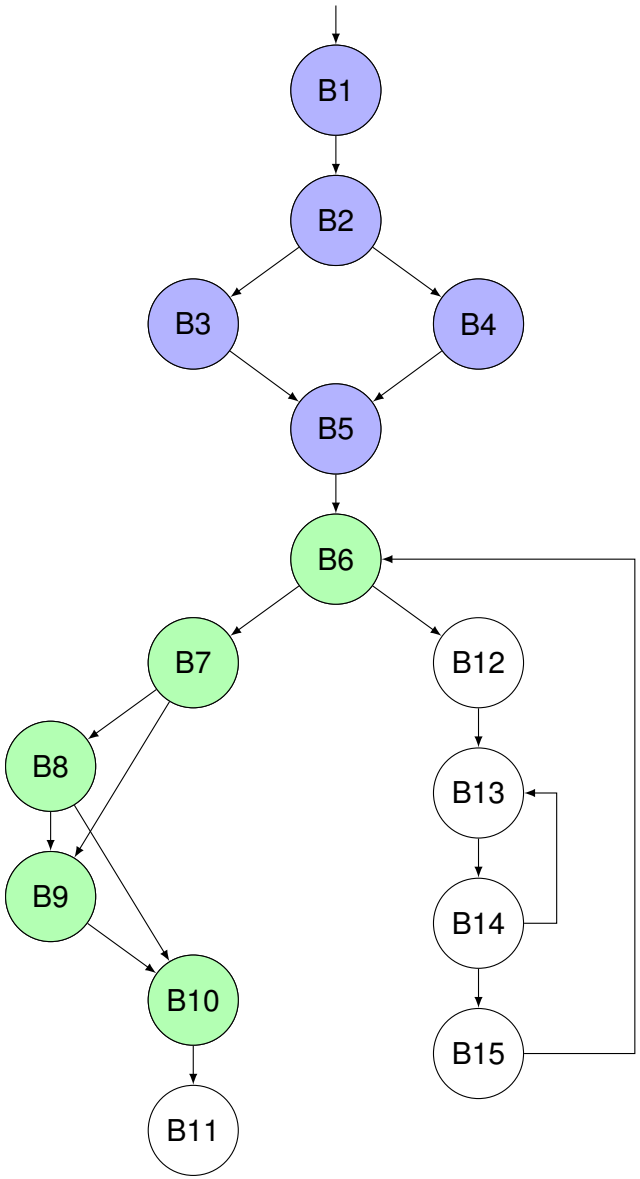
Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9\}$

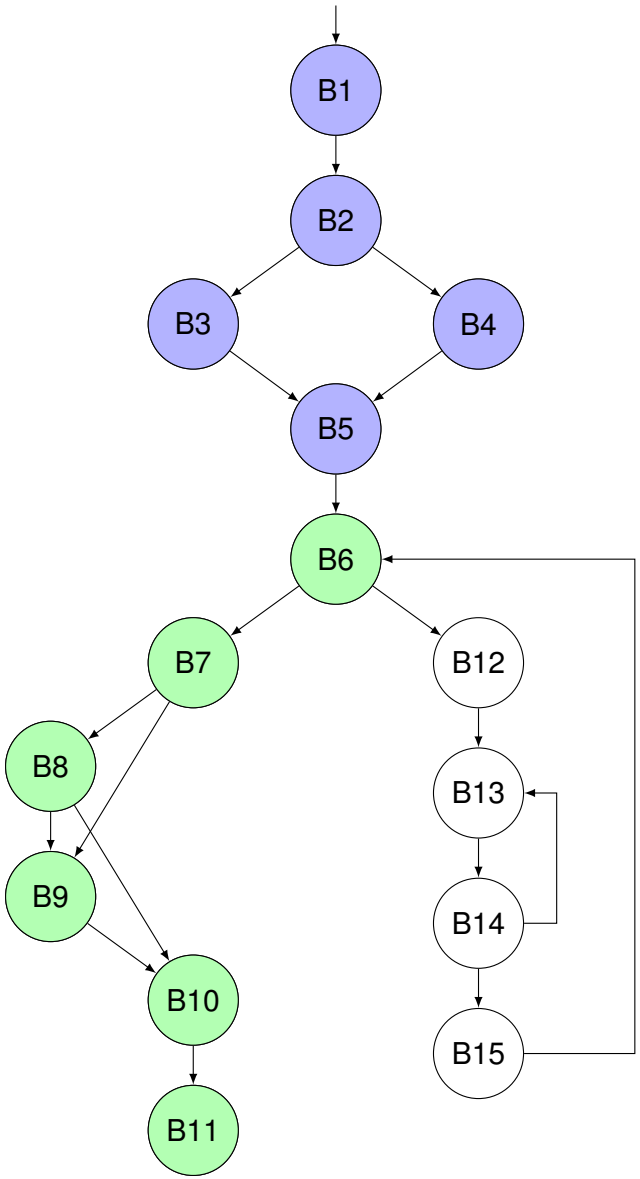
Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9, B10\}$

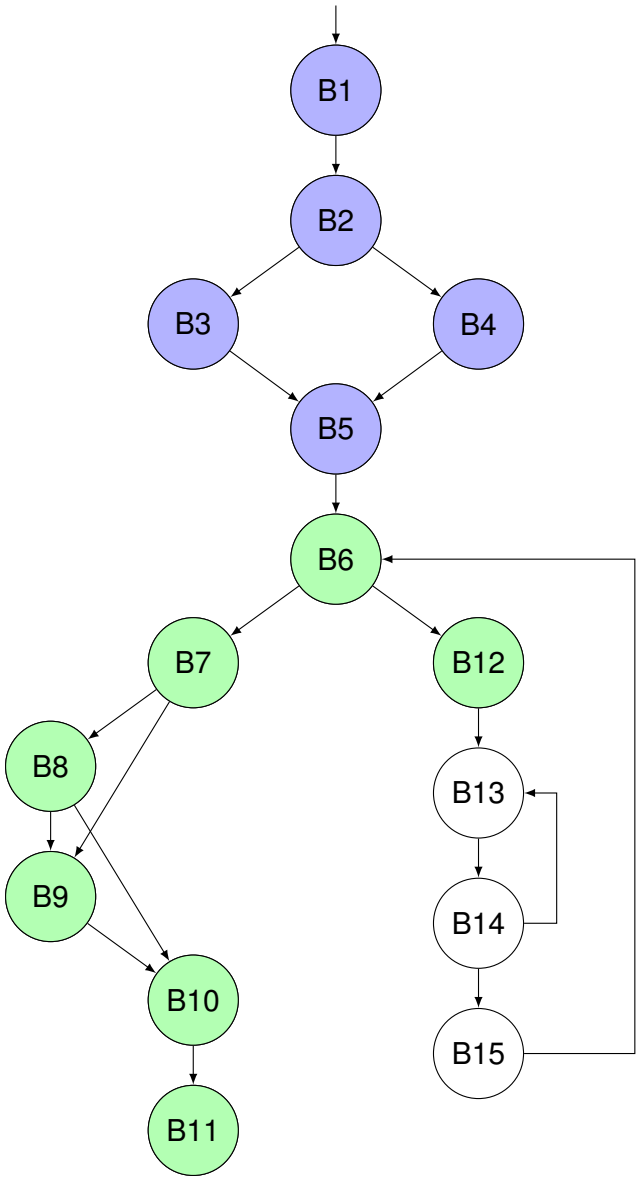
Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9, B10, B11\}$

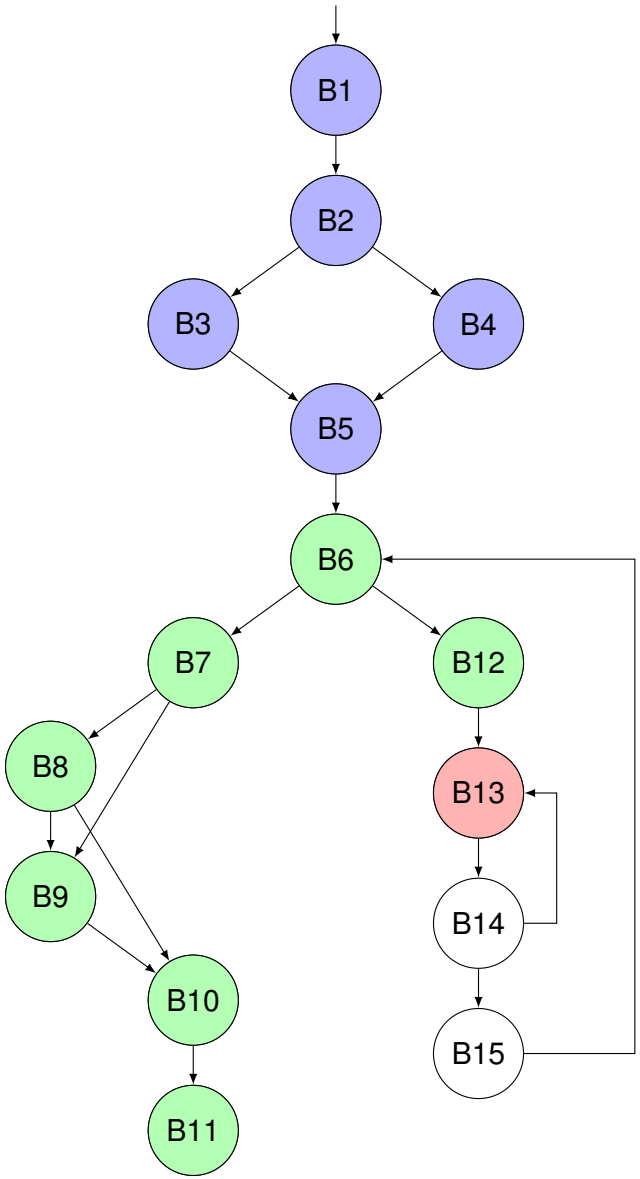
Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9, B10, B11, B12\}$

Intervallum analízis szemléltetése



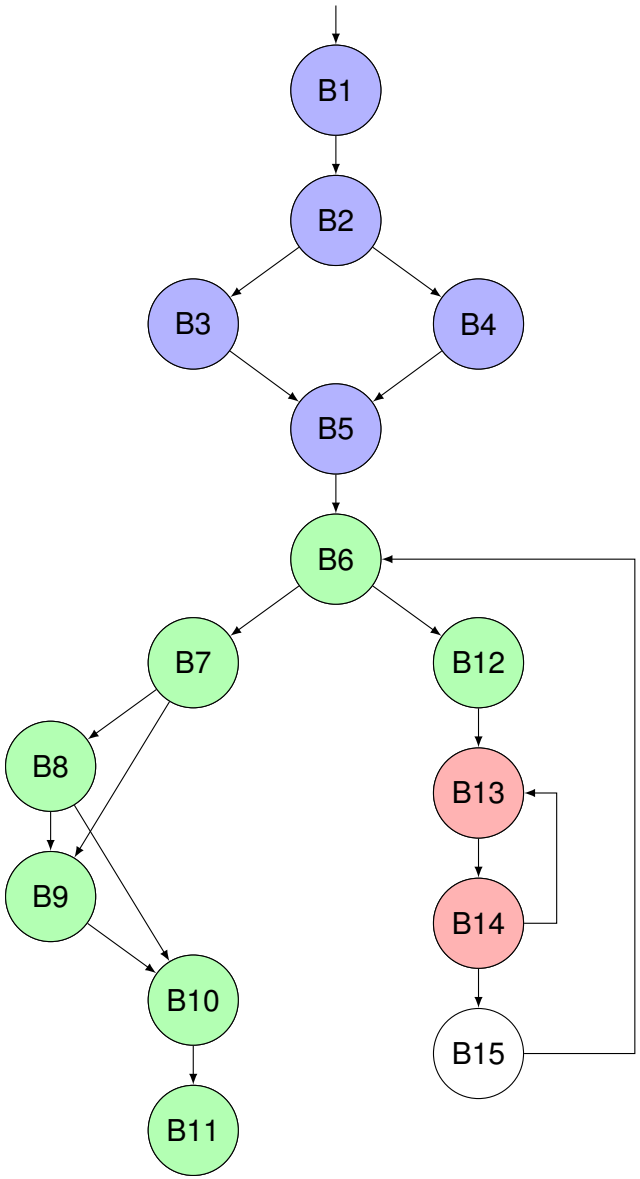
$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9, B10, B11, B12\}$

$H_3 = \{B6\}$ $I_3 = \{B13\}$

$B14$ pont nincs az I_2-es halmazban
ezért új intervallumot kell létrehozni!

Intervallum analízis szemléltetése

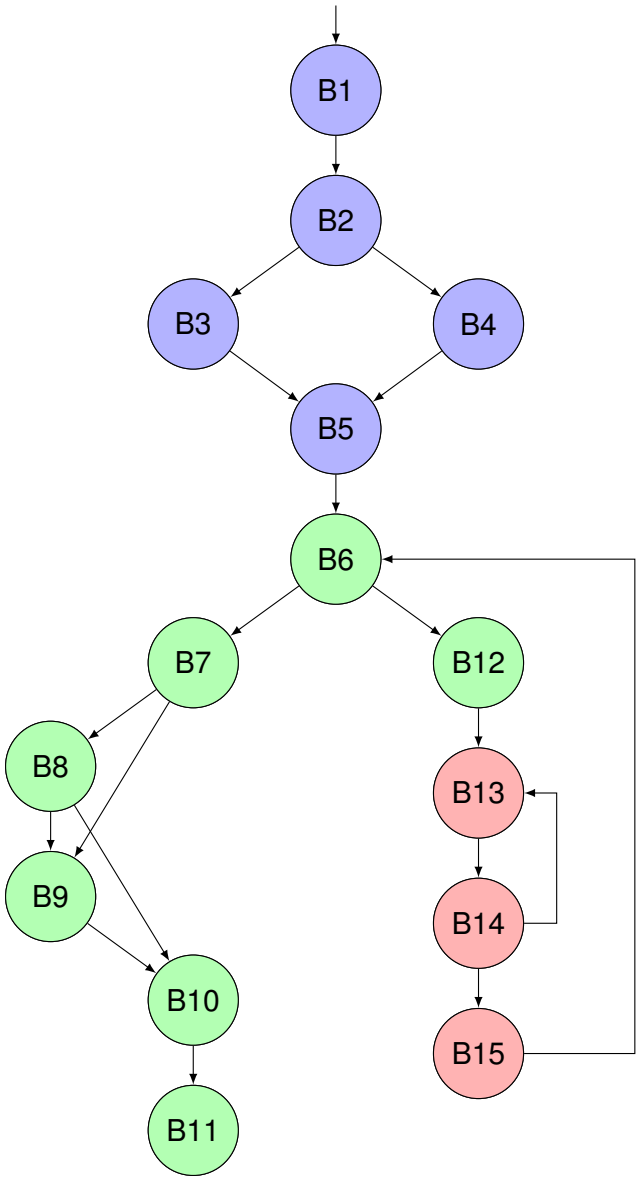


$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9, B10, B11, B12\}$

$H_3 = \{B6\}$ $I_3 = \{B13, B14\}$

Intervallum analízis szemléltetése



$H_1 = \{B1\}$ $I_1 = \{B1, B2, B3, B4, B5\}$

$H_2 = \{B6\}$ $I_2 = \{B6, B7, B8, B9, B10, B11, B12\}$

$H_3 = \{B6\}$ $I_3 = \{B13, B14, B15\}$

Improper struktúrák kezelése

Fubo Zhang és Erik H. D'Hollander bebizonyították, hogy az általuk definiált három elemi programtranszformációs lépés

- Forward Copy,
- Backward Copy,
- Cut

segítségével minden nem strukturált programrész strukturált formává alakítható. Módszerüket magasszintű blokk strukturált programozási nyelvre alkalmazták.

Haicheng Wu és társai kisebb átalakításokkal a módszert assembly szintű programra is alkalmazhatóvá tették.

Cikkek:

Fubo Zhang and Erik H. D'Hollander. Using hammock graphs to structure programs. IEEE Trans. Software Eng., 30(4):231-245, 2004.

Haicheng Wu, Gregory Diamos, Jin Wang, Si Li, and Sudhakar Yalamanchili. Characterization and transformation of unstructured control flow in bulk synchronous gpu applications. Int. J. High Perform. Comput. Appl., 26(2):170-185, May 2012.

Forward Copy

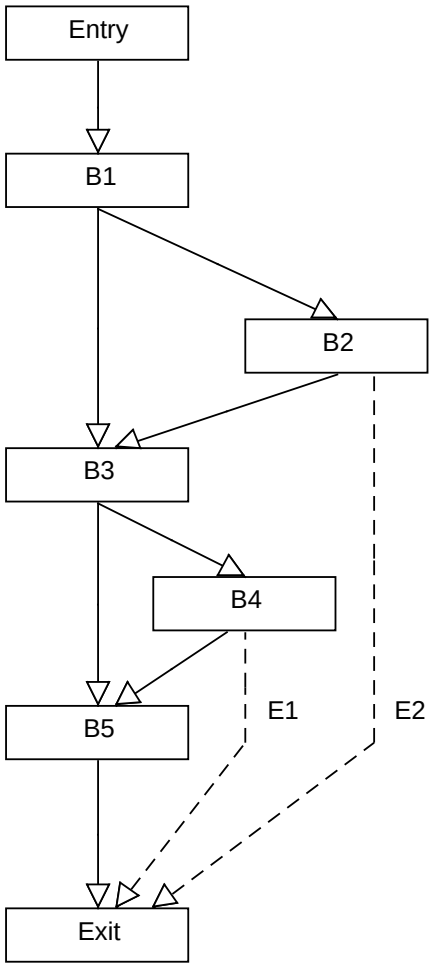
A következő ábrán a $\{B1, B2\}$ és a $\{B3, B4\}$ blokkok IfThen szerkezetet alkotnának, ha az $E1$ és $E2$ szaggatott élek nem szerepelnének.

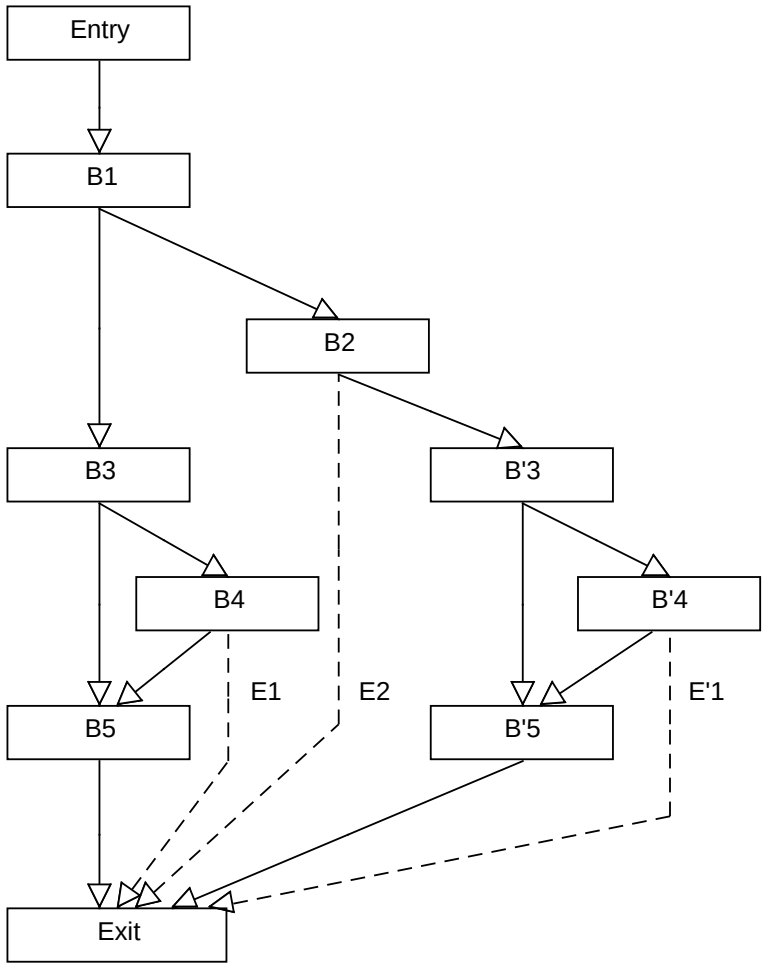
Forward Copy transzformáció:

1. *Forward copy* transzformáció során az IfThen szerkezetet alkotó $\{B1, B2\}$ részgráf közös pontja $B3$ és az $E2$ él között található gráfot le kell másolni.
2. A $(B2, B3)$ él végpontját a megismételt struktúra $B'3$ pontjára állítani.

Az így kapott jobb oldali ábra $\{B3, B4\}$ szerkezetére a forward copy transzformációt megismételve a gráf reducibilissé válik.

Forward Copy transzformáció





Backward Copy

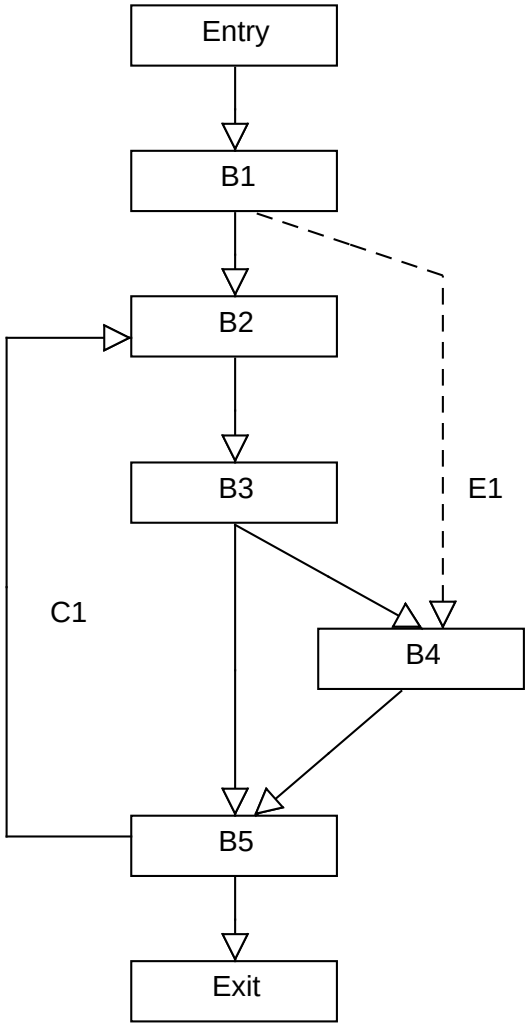
A következő ábrán található ciklus improper, mivel a belsejébe a $B1$ blokkokból az $E1$ élen keresztül be tudunk ugrani.

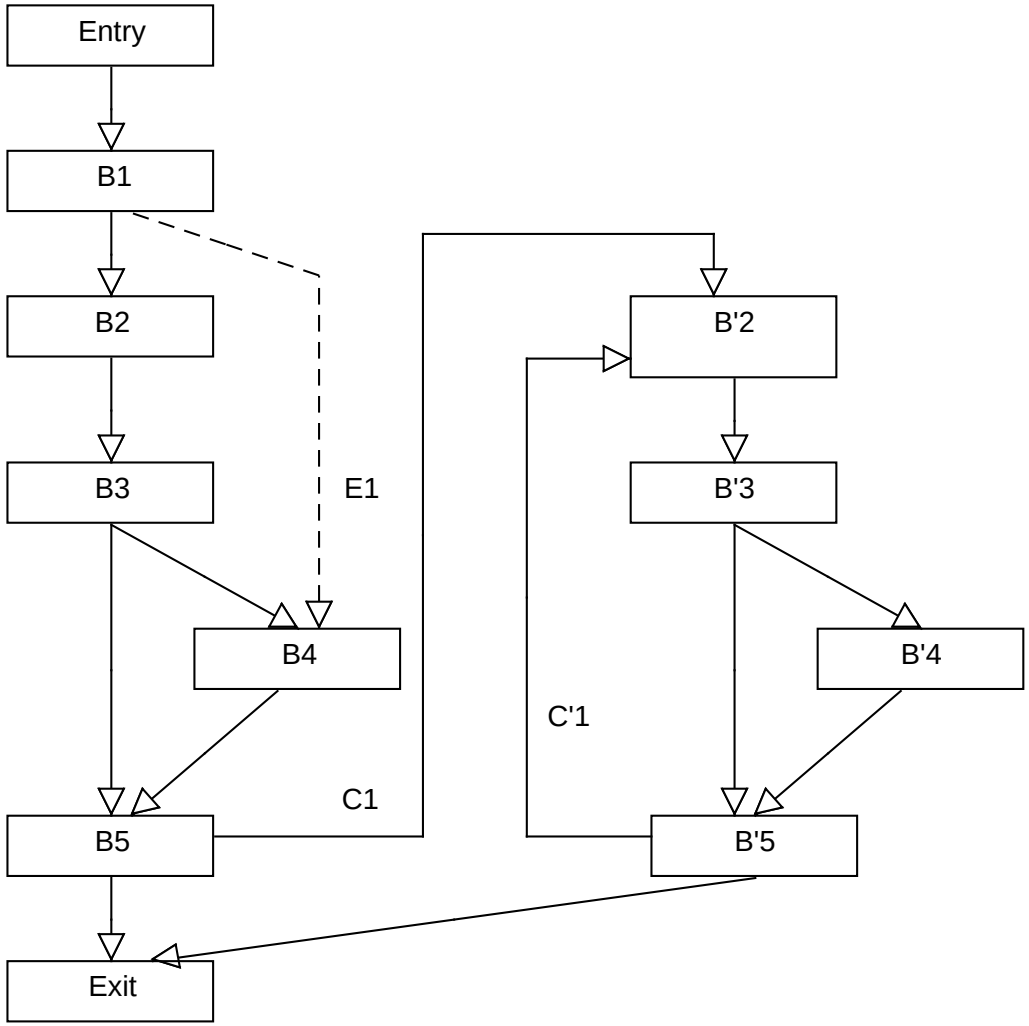
Backward Copy transzformáció:

1. A ciklust alkotó blokkokat meg kell ismételni (backward copy). Ezzel megszüntethetjük a ciklus belsejébe történő ugrást.
2. A $C1$ élet a megismételt ciklusra kell állítani.

A struktúra továbbra is irreducibilis maradt, mivel a $\{B1, B2, B3\}$ IfThen szerkezetnek két kilépési pontja van, de ezt már az előbb ismertetett forward copy transzformációval megszüntethetjük.

Backward Copy transzformáció



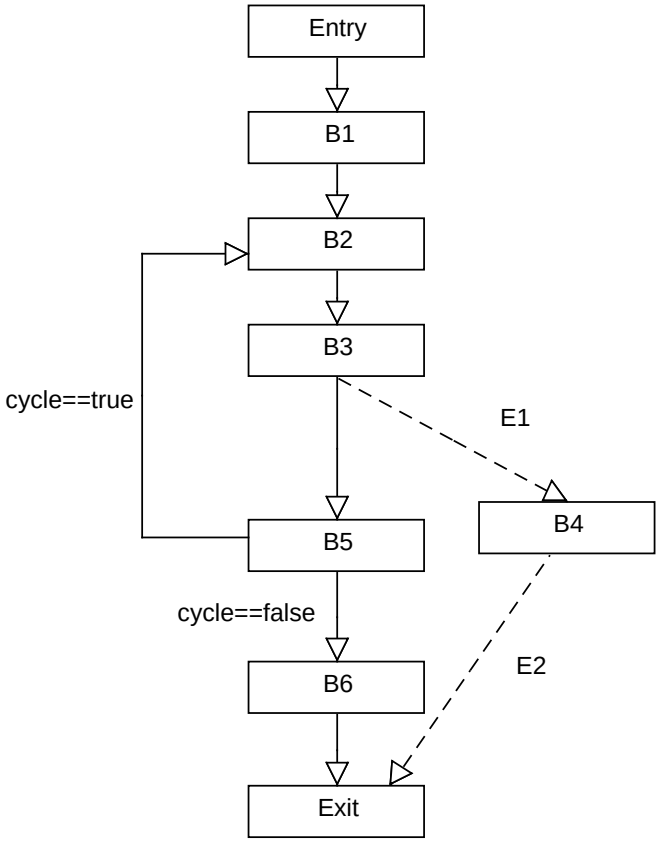


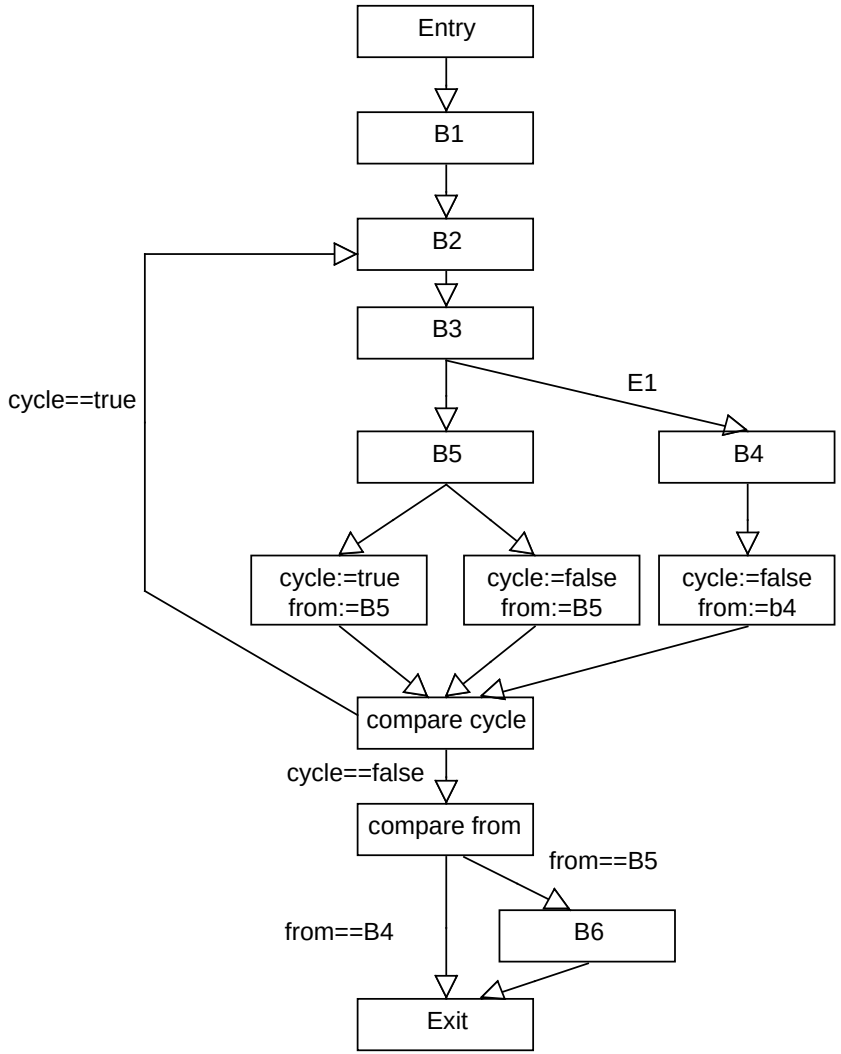
Cut

A következő ábrán a ciklus belsejéből az $E1$ és az $E2$ éleken keresztül is kiléphetünk, tehát ciklust több kilépési ponttal is rendelkezik.

1. Ciklus végét és a kilépés okát tartalmazó változók bevezetése.
2. *"compare cycle"* blokk bevezetése. A ciklus szervezését erre kell építeni.
3. *"compare from"* blokk bevezetése. Kilépés okának megfelelő folytatás elkészítése.

Cut transzformáció





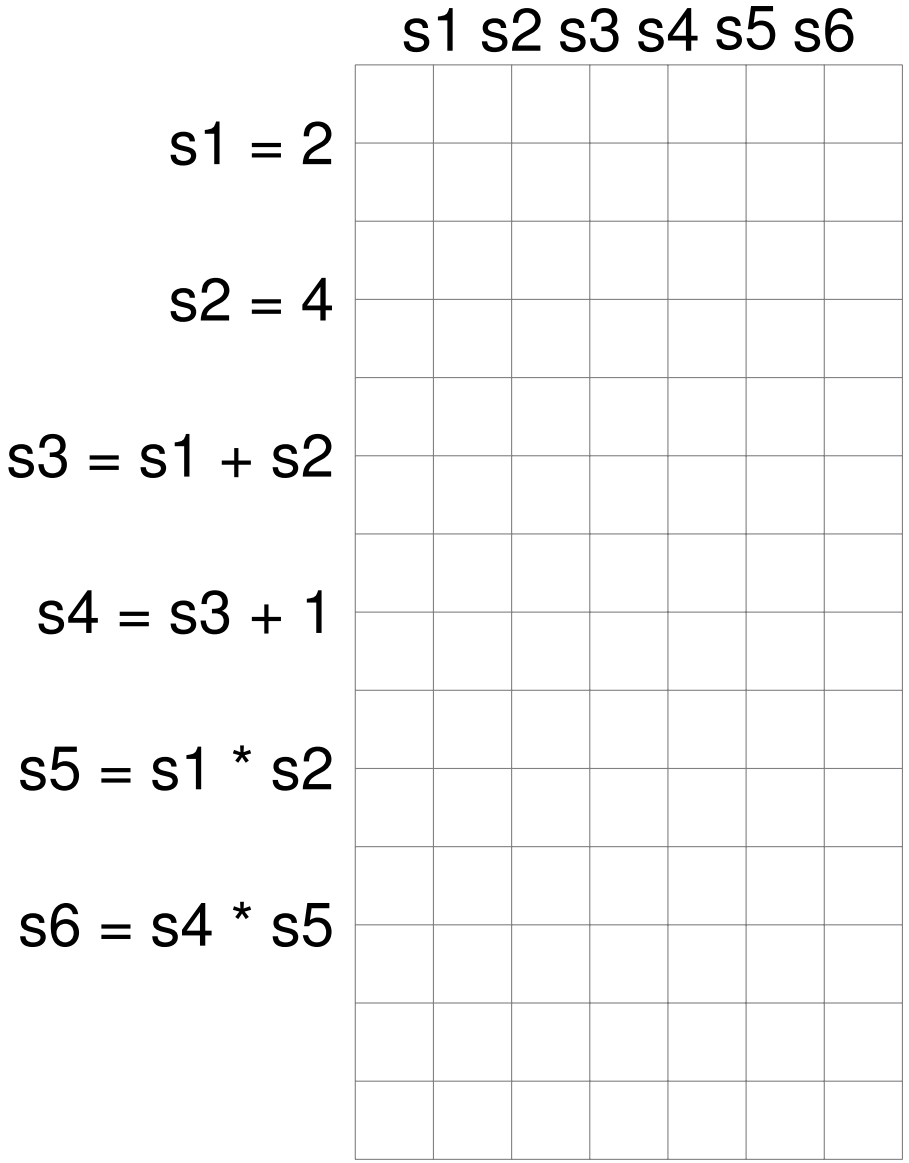
Gondolatok a strukturált programozásról

- improper — keresztél — irreducibilis — nem struktúrált
- Dahl – Dijkstra – Hoare , Strukturált programozás 7. fejezet, A programok megértéséről
- `www.hit.bme.hu/~izso/struct_prog.pdf`
- Donald E. Knuth, Stuctured Programming with goto Statesments
- Linux: Using goto In Kernel Code
- C-ben a break, continue valóban megsérti a strukturált programozás elvét, avagy irreducibilis lesz a CFG (vezérlésfolyan gráf) ?

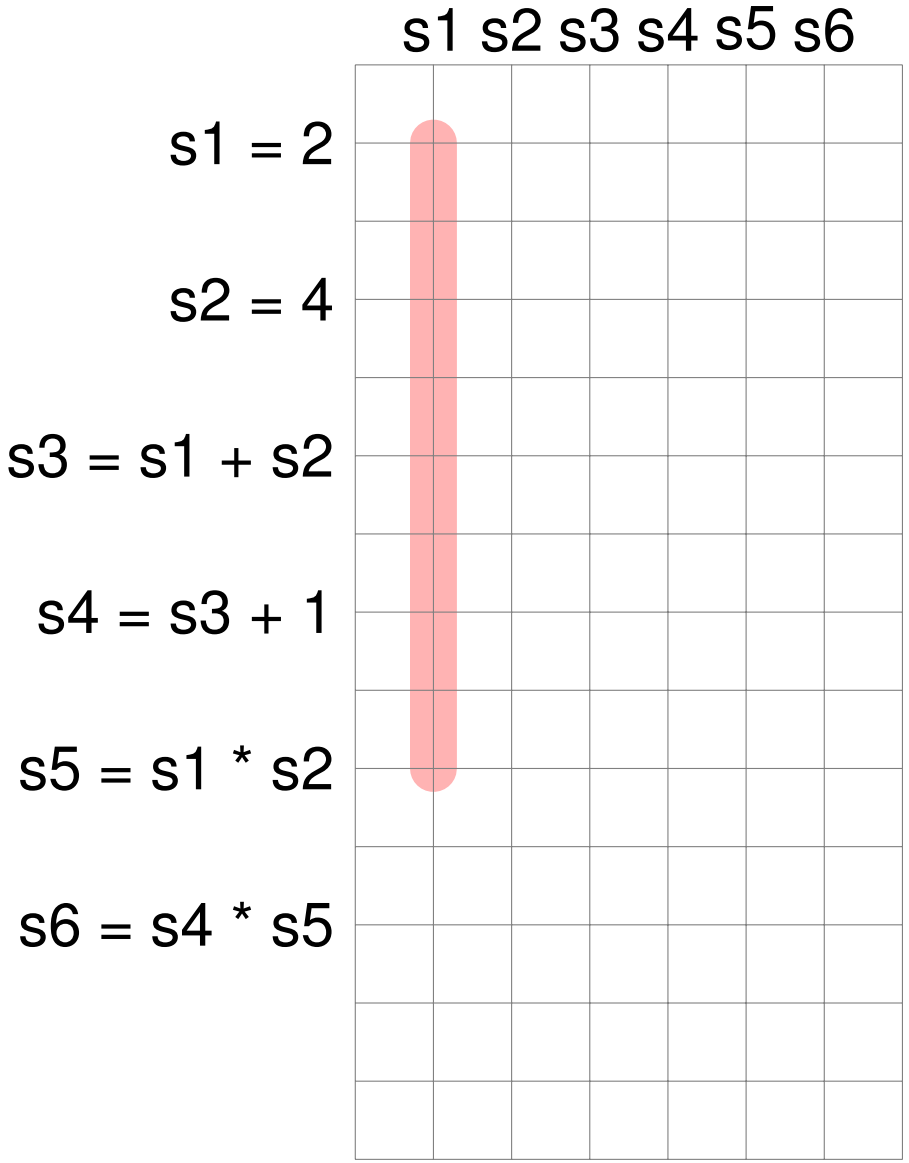
Regiszter hozzárendelés

- Intel x86 processzor utasításkészlete 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 az egyidejűleg használt regiszterek számát;
- általában színezési problémára vezethető vissza.

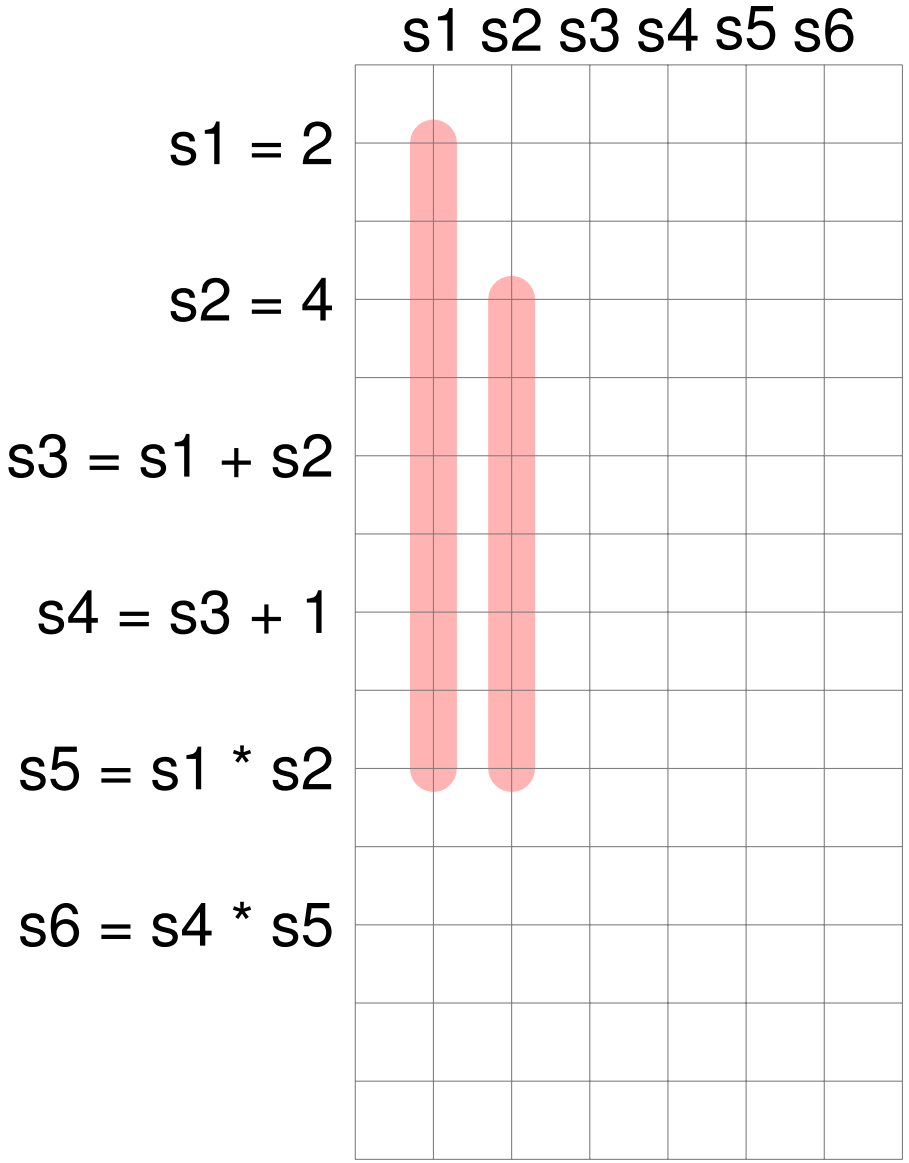
Regiszter hozzárendelés



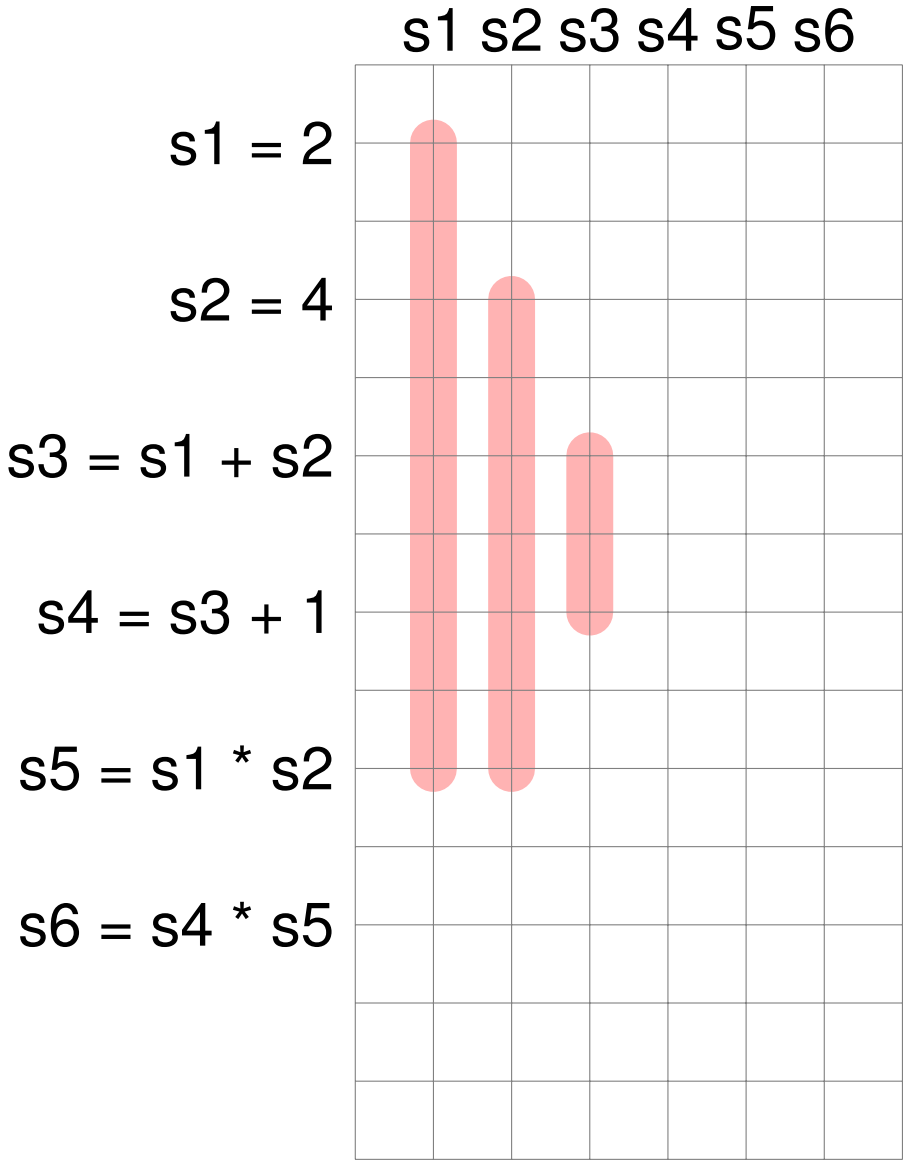
Regiszter hozzárendelés



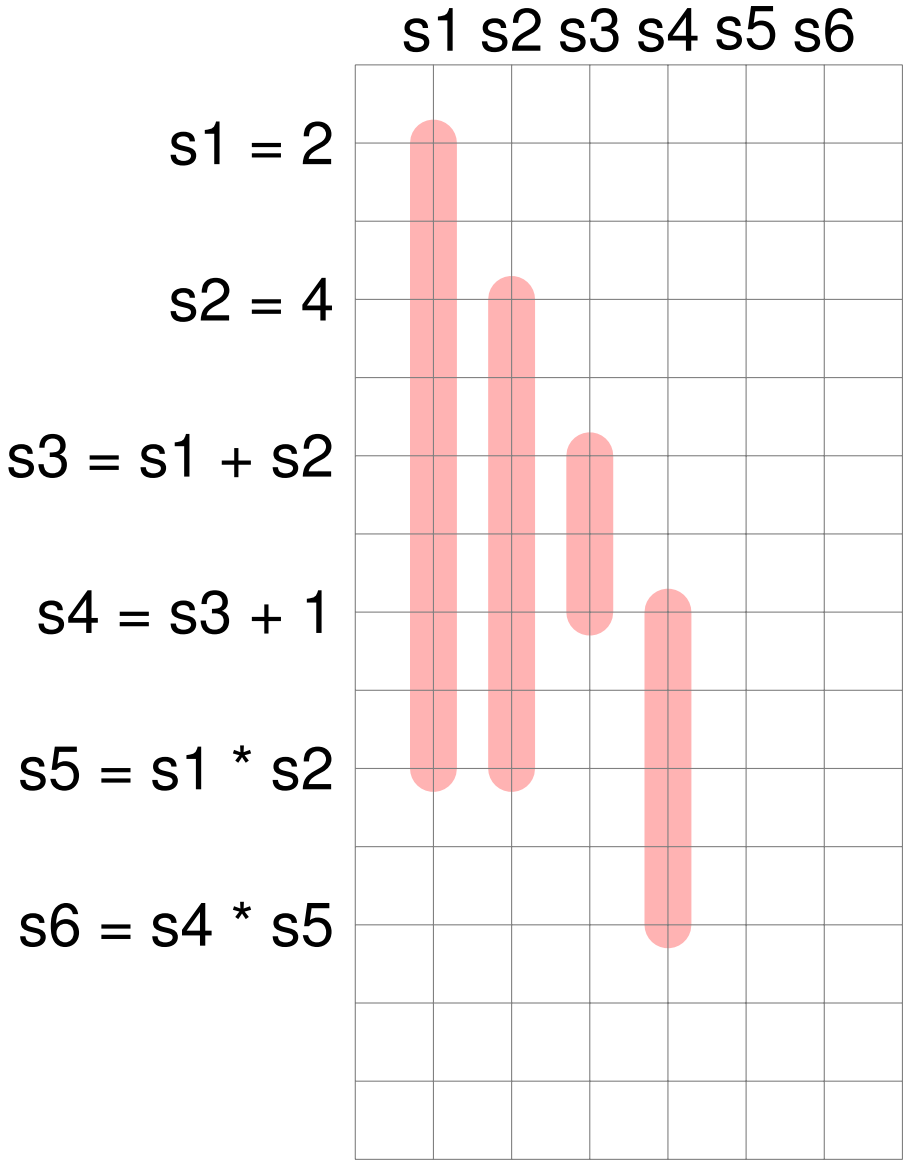
Regiszter hozzárendelés



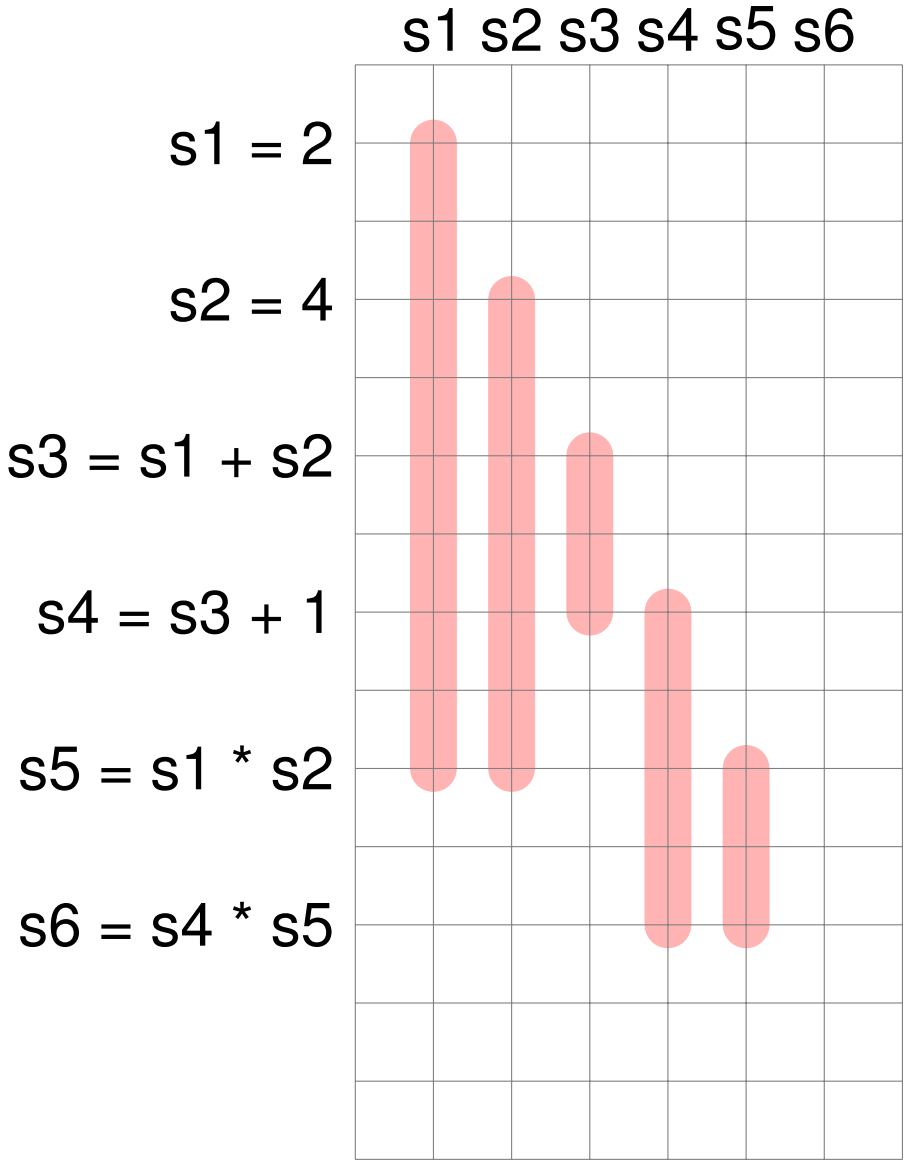
Regiszter hozzárendelés



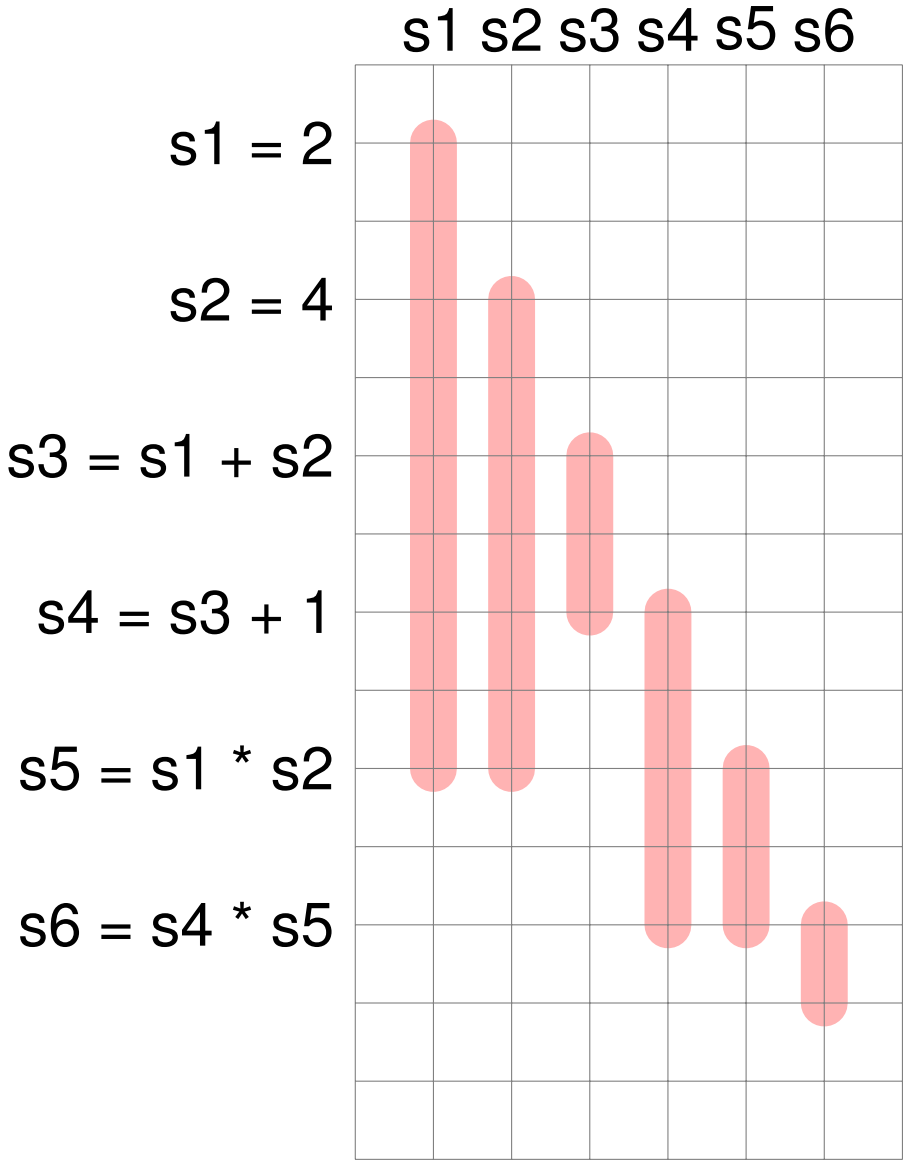
Regiszter hozzárendelés



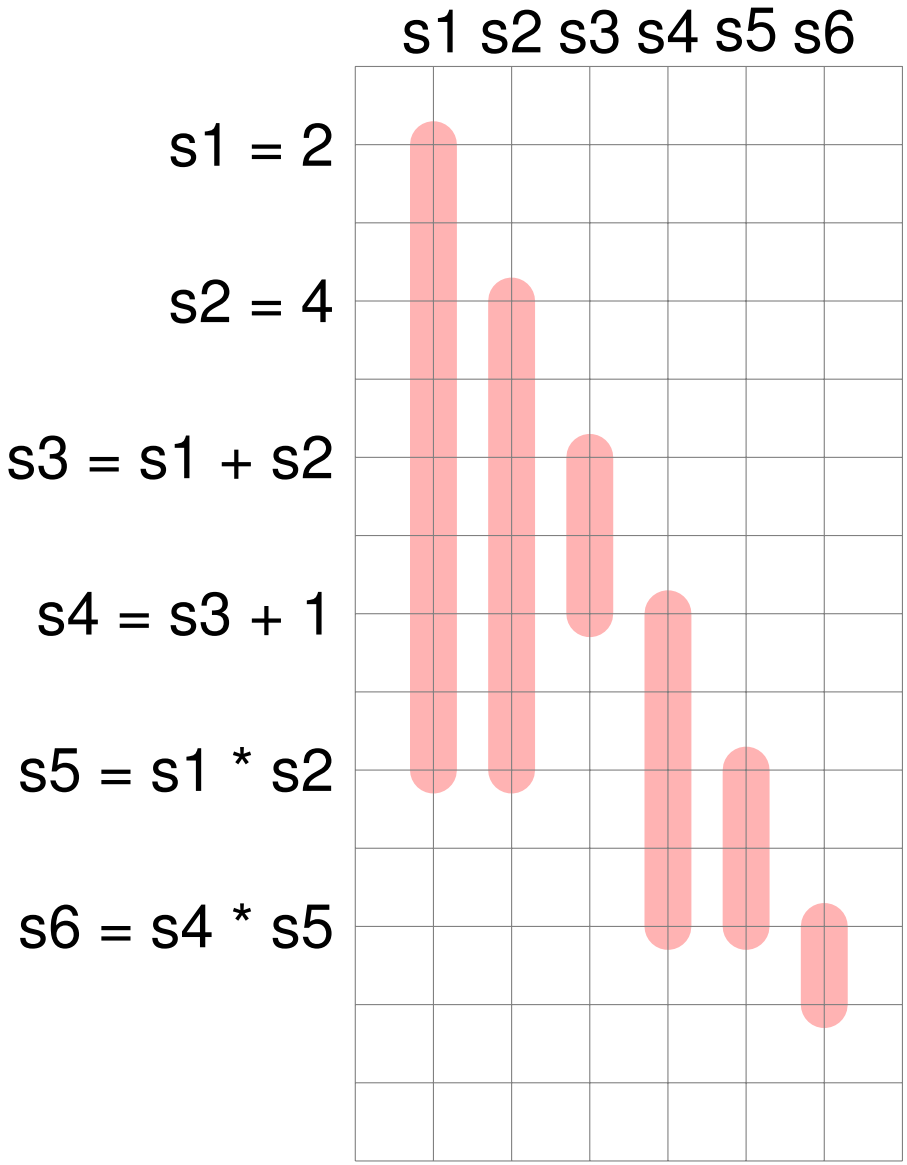
Regiszter hozzárendelés



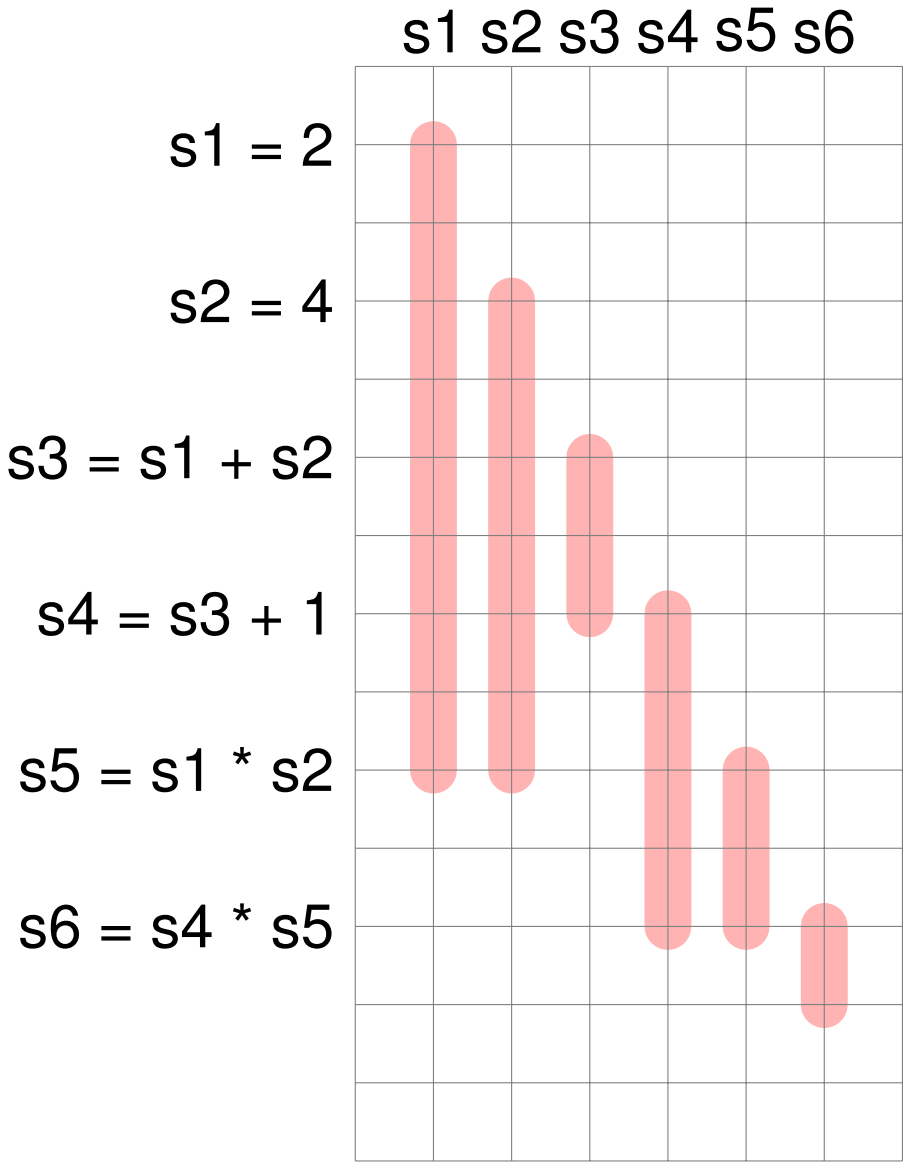
Regiszter hozzárendelés

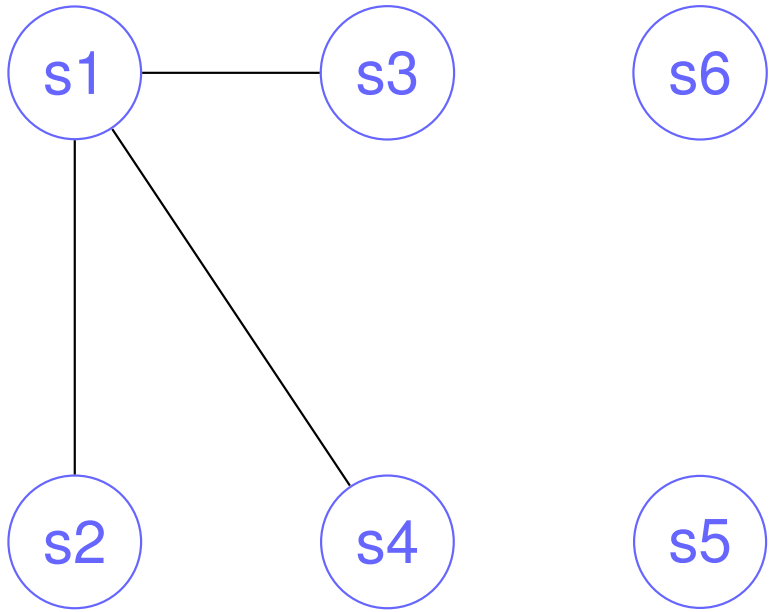


Regiszter hozzárendelés

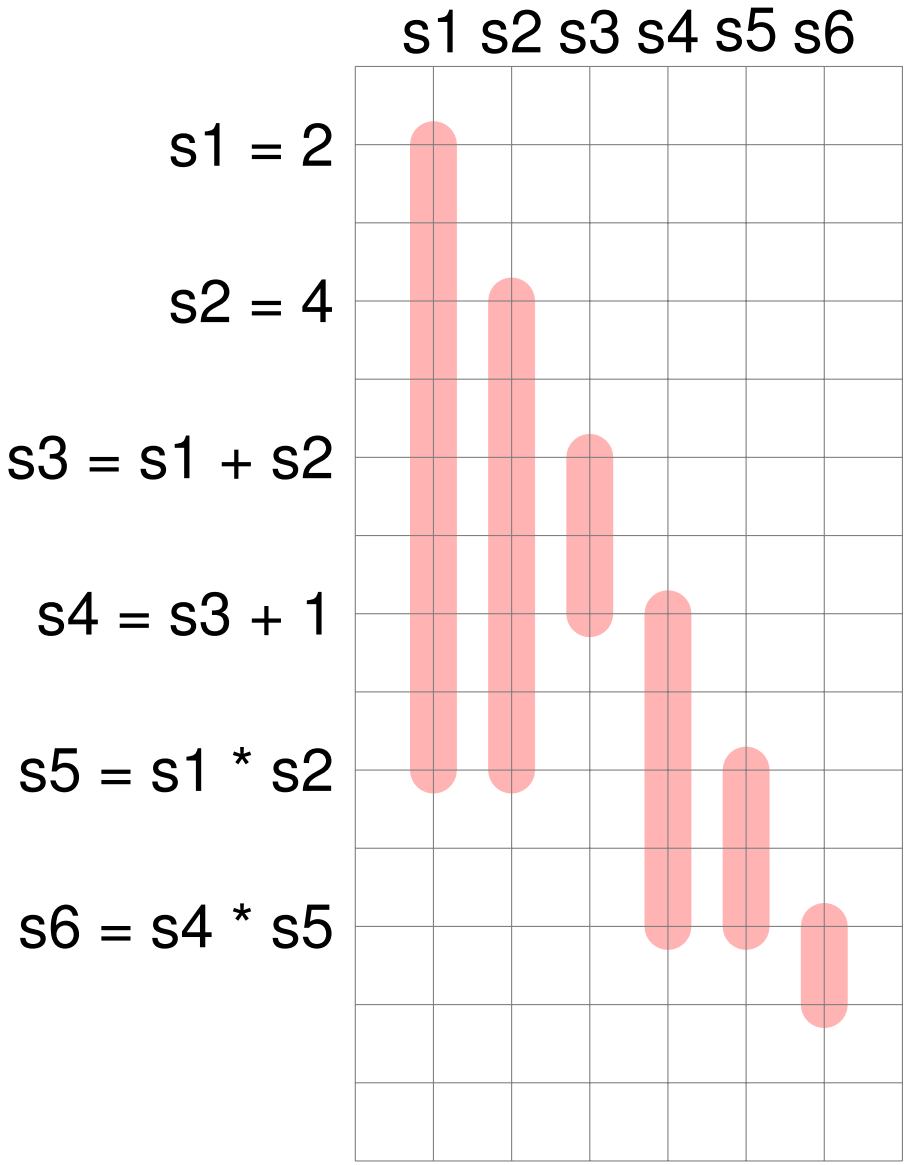


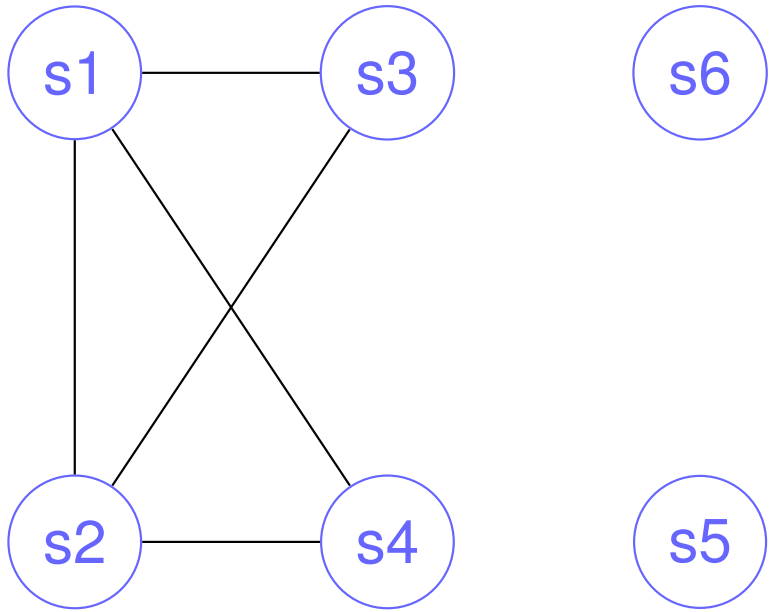
Regiszter hozzárendelés



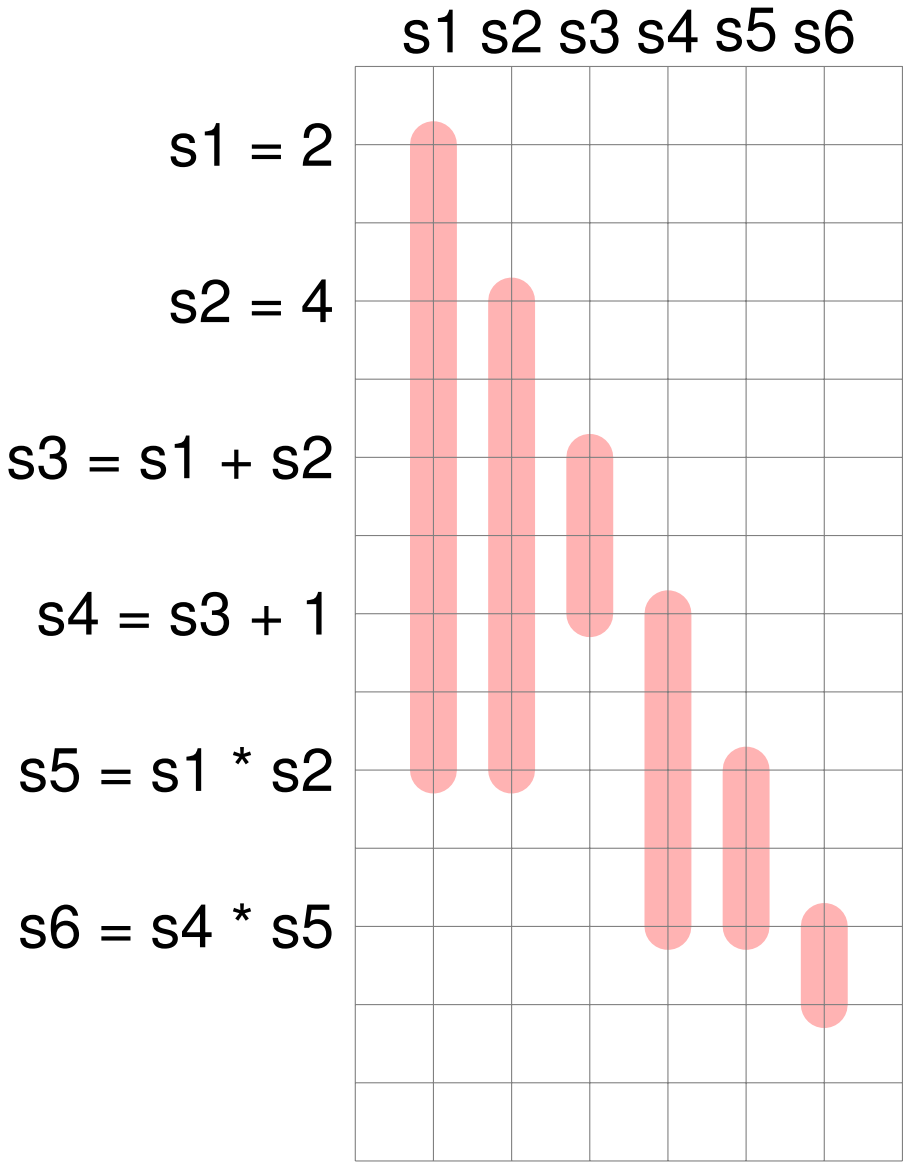


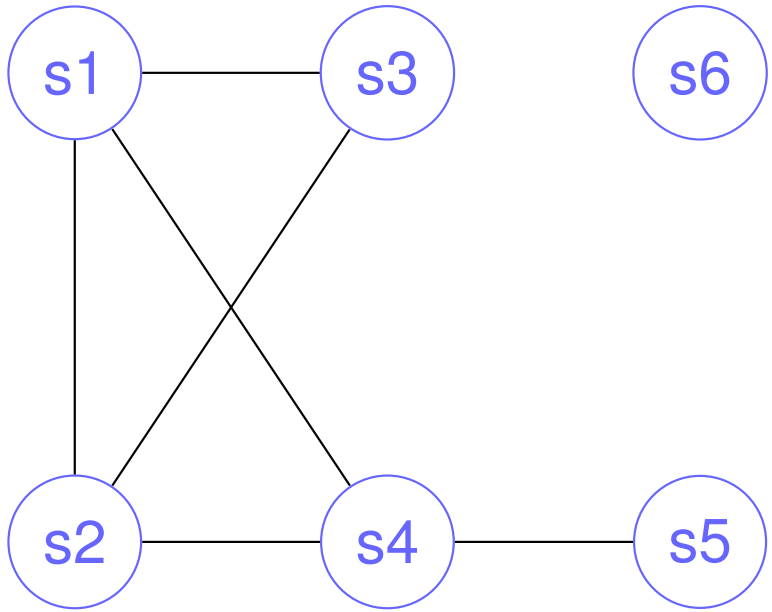
Regiszter hozzárendelés





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 1

Intel mikroprocesszor története

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;
- 24-bit címsín;
- 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 2

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;
  - művelet 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

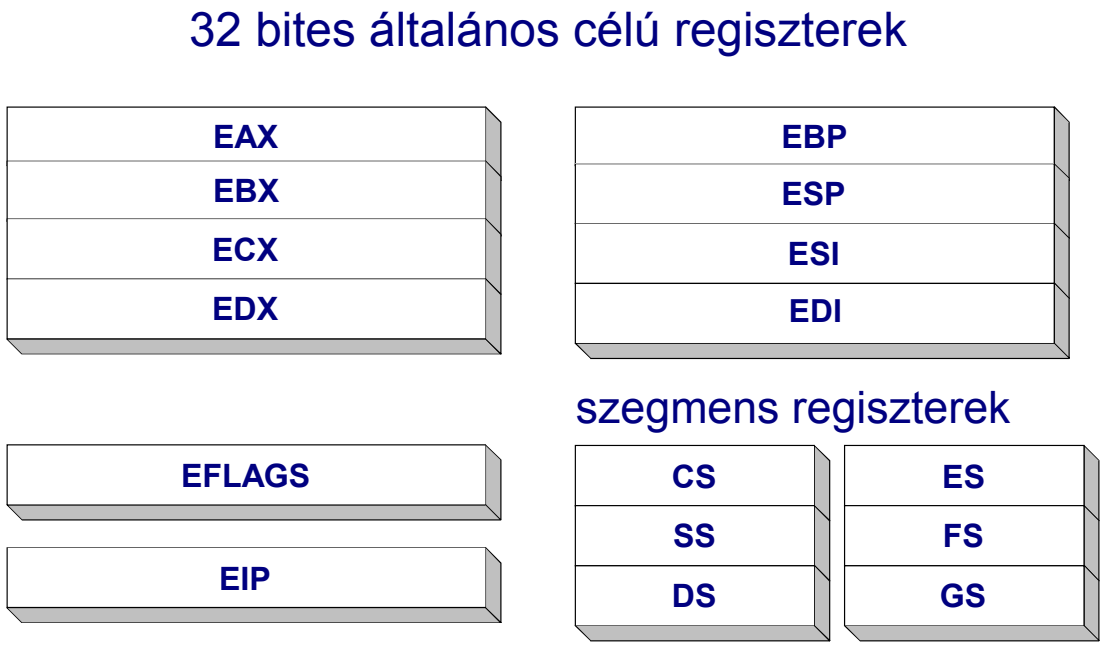
- Utasításkészlete CISC tulajdonságot mutat.
- Belül a CISC utasításokat egyszerű mikróműveletekre bontja (μ Op) és párhuzamosan képes ezeket végrehajtani. (pl. Sandy Bridge Pipeline architektúra).

Section 3

Intel X86 utasításkészlet

X86 regiszterek

- 8 darab 32 bites regiszter;
- 6 darab szegmens regiszter;
- statusz regiszter EFLAGS
- utasításszámláló EIP

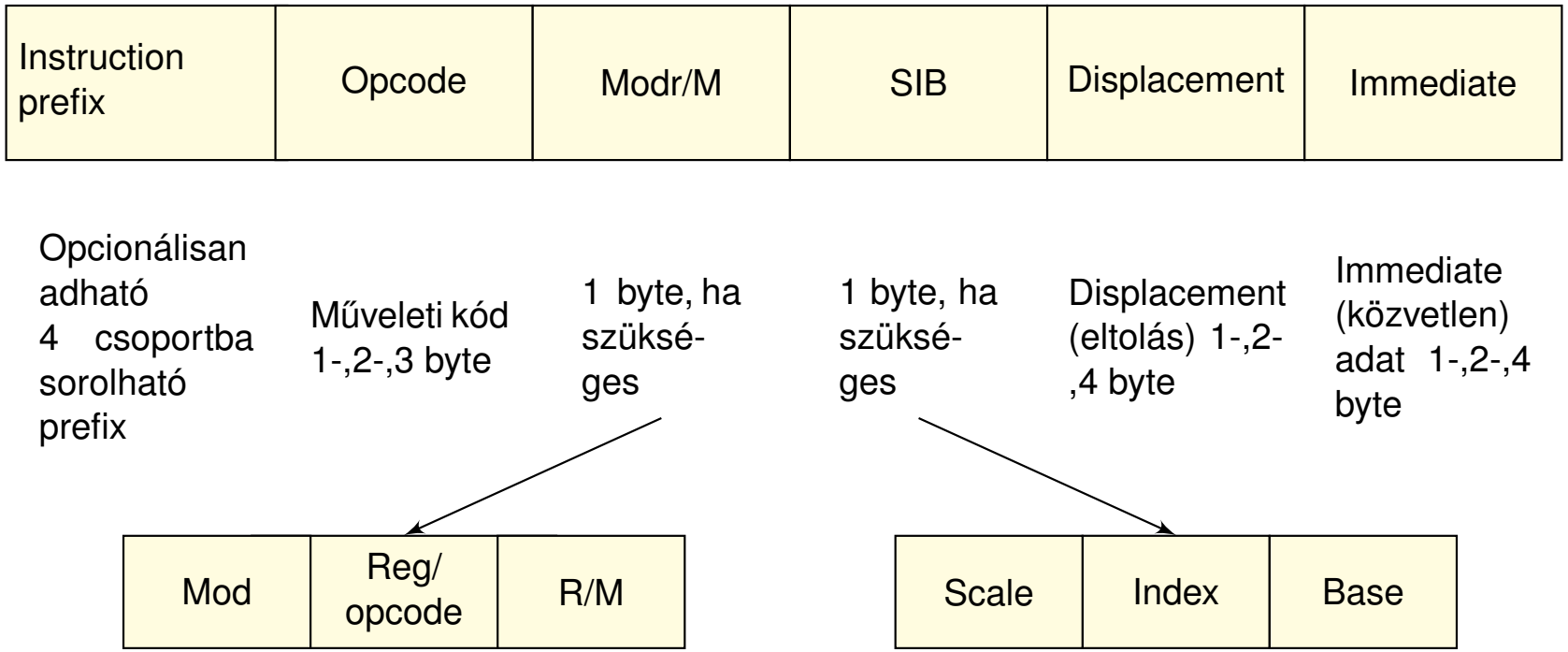


Regiszterek speciális szerepe egyes utasításokban

Speciális de nem kizárólagos felhasználása a regiszteknek.

- **EAX** akkumulátor regiszter, szorzáshoz osztáshoz;
- **ECX** counter regiszter (ciklusszámlálásra);
- **ESP** stack pointer;
- **ESI** string műveletek esetén a forrás memóriaterületet indexeli;
- **EDI** string műveletek esetén a cél memóriaterületet indexeli;
- **EBP** bázis pointer a stack kezeléshez.

x86 utasítás formátum



Utasítás prefix

1. csoport lock vagy repeat prefix
   - F0H LOCK prefix multiprocesszoros környezetben az osztott memóriához kizárólagos hozzáférést biztosít.
   - F2H REPNE/REPNZ prefix, amely string vagy input/output utasításokhoz lehet használni.
   - F3H REP vagy REPE/REPZ prefix, amely string vagy input/output utasításokhoz lehet használni.
2. operandusok méretének az átdefiniálása
   - csoport 66H operandus méretének megváltoztatása, azaz a 16 és 32 bites operandusok között választhatunk.
   - csoport 67H cím méretének a megváltoztatása.

Utasítás prefix

3. szegmens módosítás, amely ugró utasításokra nem érvényes
   - 2EH CS
   - 36H SS
   - 3EH DS
   - 26H ES
   - 64H FS
   - 65H GS
4. elágazásbecslés feltételes vezérlésátadó utasításokhoz
   - 2EH valószínűleg az ugrás nem fog végrehajtódni;
   - 3EH valószínűleg az ugrás be fog következni.