Scheduling

Rozvrhování instrukcí

Architektura back-endu

- **ILP (instruction-level parallelism)**
  - Pipeline
    - Instrukce se zpracovává v několika fázích (stages)
      - fetch – read – execute – write
    - V jednom okamžiku se provádějí různé fáze různých instrukcí
    - Závislosti po sobě jdoucích instrukcí způsobují:
      - Pipeline stall – zdržení (i486)
      - Nekorektní provedení kódu (Sparc)
  - Superskalární procesory (MIMD)
    - Několik výkonných jednotek (Pentium: 2)
    - V jednom okamžiku se provádějí stejné fáze několika instrukcí
  - Vektorové procesory (SIMD)
    - Mnoho výkonných jednotek (Cray: 64)
      - Slabší varianty: Pentium MMX/SSE
    - V jednom okamžiku se provádí tatáž instrukce mnohokrát
    - Využití nepatří pod pojem scheduling
      - Automatická vektorizace používá podobné algoritmy jako některé metody schedulingu

Scheduling

- **Provádí se na jednom základním bloku**
  - Trace scheduling: Vybraná posloupnost BB slita do jednoho
  - Software pipelining: Speciální řešení pro BB jako cyklus
- **Vstup**
  - Dag
    - Uzly = instrukce
    - Hrany = závislosti
  - Model procesoru
    - Latence – časování závislých dvojic instrukcí
    - Rezervační tabulky – schopnosti paralelního zpracování
- **Výstup**
  - Přiřazení času každému uzlu dagu
    - Čas měřen cykly procesoru
    - Instrukce trvá několik cyklů – zvolen referenční bod
      - Obvykle začátek zpracování po načtení a dekódování instrukce

- **Dag**
  - Uzly = instrukce
  - Hrany = závislosti
    - Dependence vzniklé z předávání dat v registrech
    - Dependence a antidependence z přístupů do paměti
      - Opatrný přístup: Možná závislost => závislost
    - Antidependence vzniklé ze soupeření o registry
      - Při schedulingu po alokaci registrů
    - Trace scheduling + software pipelining: Kontrolní dependence
      - Některé operace musejí počkat na výsledek podmíněného skoku
    - Další závislosti z technických příčin
      - Manipulace se zásobníkem apod
  - Instrukce volání procedury
    - Pro účely schedulingu obvykle považována za hranici mezi BB
      - Neumožňuje optimalizaci přesunem přes volání

Scheduling

- **Výstup**
  - Přiřazení času každému uzlu dagu
- **Aplikace výstupu schedulingu**
  - Běžné procesory (Intel IA-32, včetně x86_64)
    - Seřazení instrukcí podle schedulovaného času
      - Při shodě času nutno dodržet (anti-)dependence
    - Procesor nemusí dodržet předpokládaný čas
  - Procesory se sekvenčními body (Intel IA-64)
    - V kódu jsou mezi instrukcemi označeny sekvenční body (stops)
      - Procesor má právo přeházet pořadí instrukcí mezi sekvenčními body
      - Ignorují se antidependence i některé dependence
    - Výstupem scheduleru jsou i sekvenční body
  - VLIW procesory
    - Very Large Instruction Word
      - Instrukce řídí paralelní činnost jednotek procesoru
    - Jeden schedulovaný čas = jedna instrukce

- **Scheduling pouze odhaduje skutečné časování**
  - Skutečné časování je ovlivněno nepředvídatelnými jevy
    - Zbytky rozpracovaných instrukcí z předchozího BB
      - Řešení: Trace-scheduling, řízení profilem
    - Paměťová hierarchie
      - Doba přístupu k paměti závisí na přítomnosti v cache
      - Obvykle se předpokládá nejlepší možný průběh
      - Speciální problém: Multithreaded aplikace na multiprocesorech
    - Fetch bandwidth
      - Instrukce nemusí být načteny a dekódovány včas
      - Zdržují skoky a soupeření o přístup do paměti
      - Přesné simulování fetch jednotky by neúměrně komplikovalo scheduler
  - Scheduler nezná skutečné závislosti přístupů do paměti
    - Musí postupovat opatrně a zohledňuje i nejisté závislosti
    - Procesor zná skutečné adresy přístupů a detekuje pouze skutečné závislosti
      - Agresivně optimalizující procesor může zvolit zcela jiné pořadí instrukcí

- **Model procesoru**
  - Latence – časování závislých dvojic instrukcí
    - Počet cyklů procesoru, který musí proběhnout mezi referenčními body závislých instrukcí
    - U antidependencí a ve speciálních případech může být nulová
    - U procesorů se sekvenčními body může být záporná
    - Latence se obvykle zapisuje ke hranám dagu
      - Přiřazena na základě klasifikace závislosti podle tabulek latencí

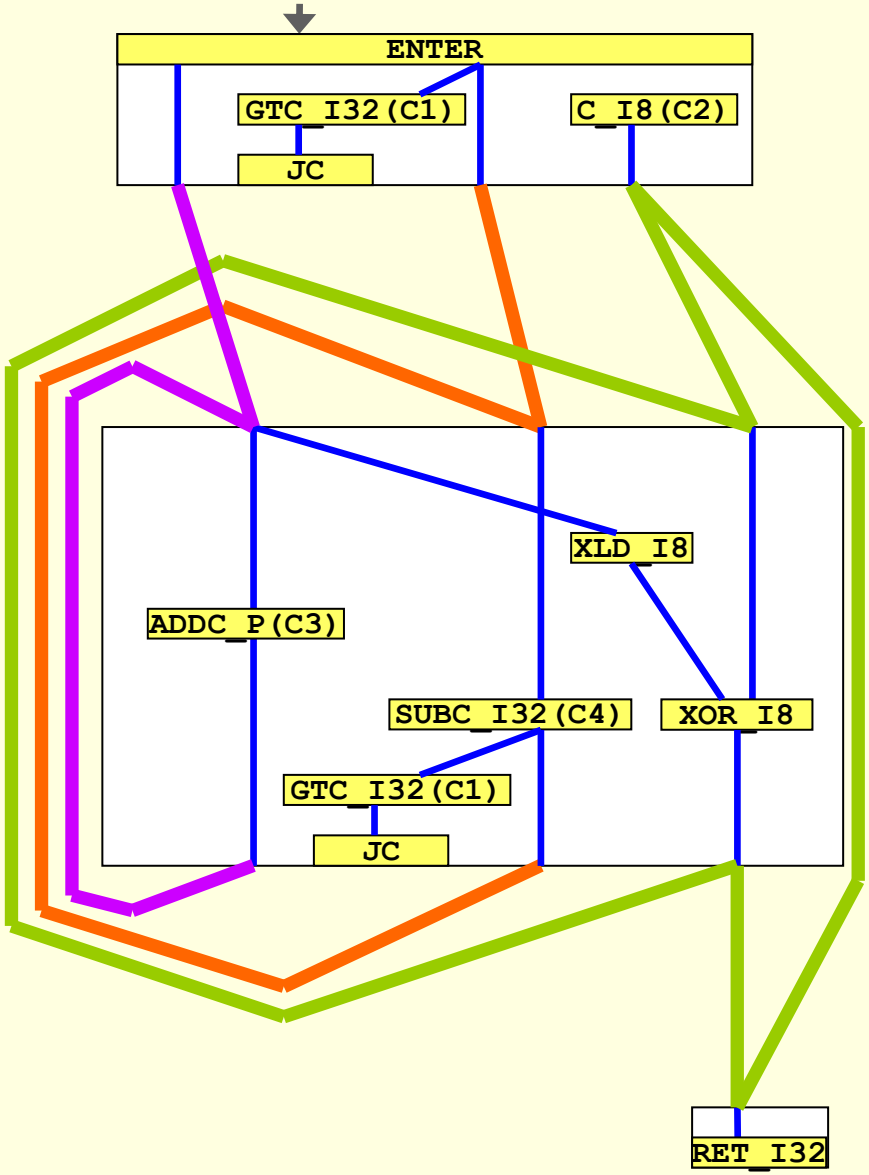
Model procesoru

Rezervační tabulky – schopnosti paralelního zpracování

- Procesor je rozdělen na funkční jednotky různých druhů
- Je určen počet jednotek každého druhu
  - Limit: Kind -> **N**
- Pro každou instrukci definována rezervační tabulka
  - Res(instr): Time × Kind → **N**
  - Počet jednotek daného druhu, který instrukce potřebuje v daném čase (měřeno od referenčního bodu)
- Rezervační tabulky jsou nutné i pro procesory, které nejsou super-skalární
  - Mají Limit(k)=1, ale různé a tudíž konfliktní rezervační tabulky

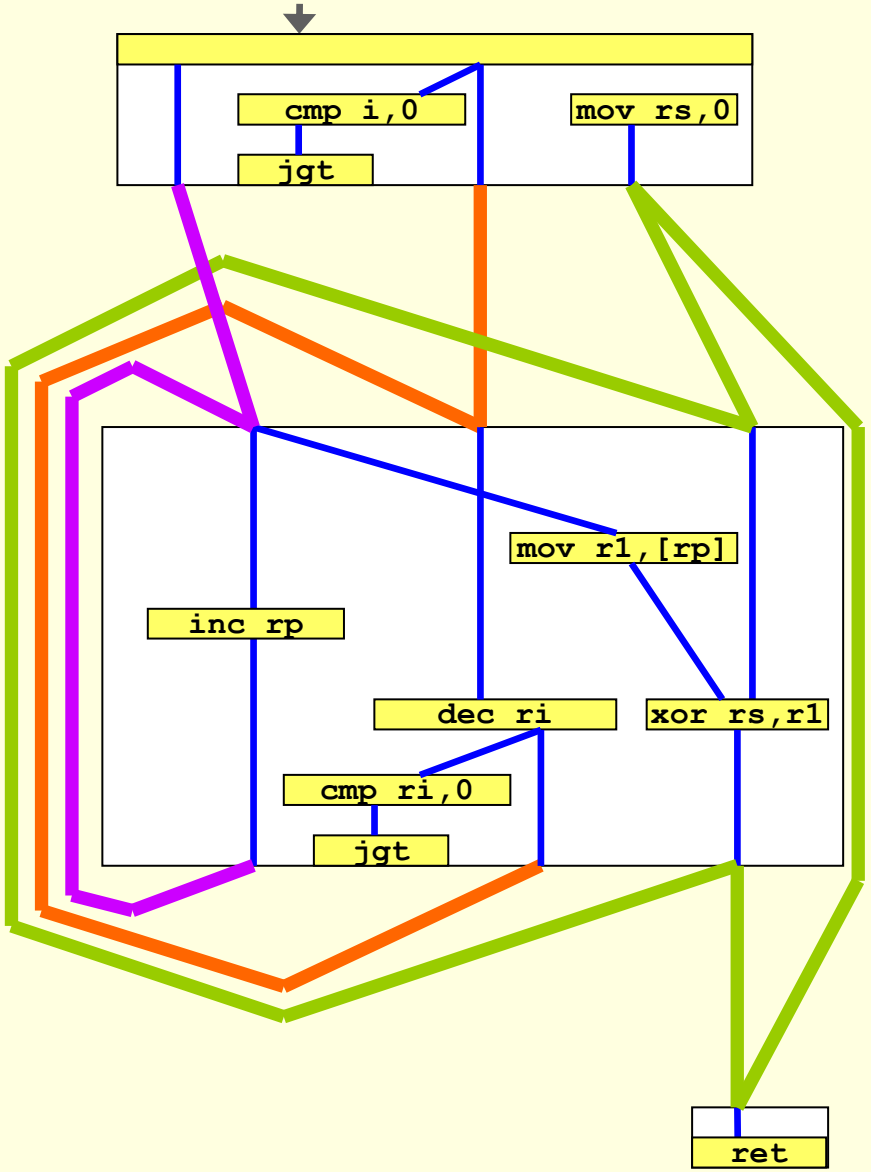
Příklad – mezikód střední úrovně


```
char chksum( char * p, int i)
{
  char s = 0;
  while ( i > 0 )
  {
    s ^= *p++;
    --i;
  }
  return s;
}
```



Příklad – mezikód nízké úrovně


```
char chksum( char * p, int i)
{
  char s = 0;
  while ( i > 0 )
  {
    s ^= *p++;
    --i;
  }
  return s;
}
```



Příklad – latence

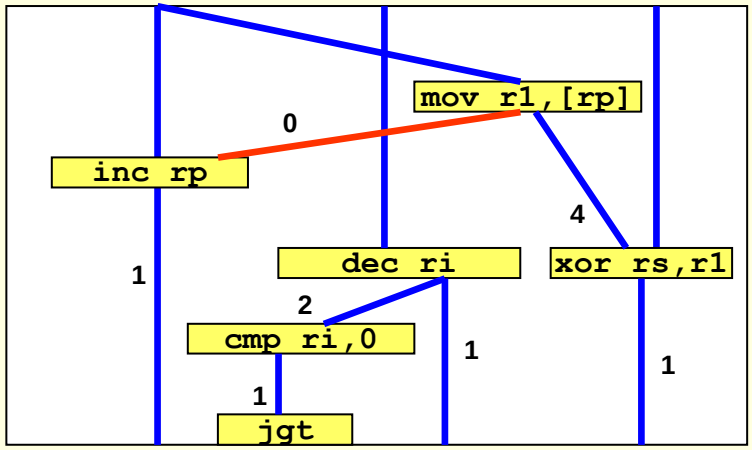
- Tabulka latencí
  - Instrukce-instrukce

z instrukce	do instrukce	čas
cmp ri,0	*jgt*	1
mov *r1*,[rp]	xor rs,*r1*	4
dec *ri*	cmp *ri*,c	2
mov r1,[*rp*]	inc *rp*	0

  - Instrukce-konec BB
    - Pesimistická varianta (instrukce musí být dokončena v tomto BB)

z instrukce	čas
inc rp	2
jgt	1
dec ri	2
xor rs,r1	2

- Pesimistický přístup na konci BB umožňuje optimistiký přístup na začátku: Latence začátek BB-instrukce jsou považovány za nulové



- Instrukce jgt musí být poslední
  - Latence vůči konci BB se normalizují odečtením latence jgt

Příklad – rezervační tabulky

Rezervační tabulky

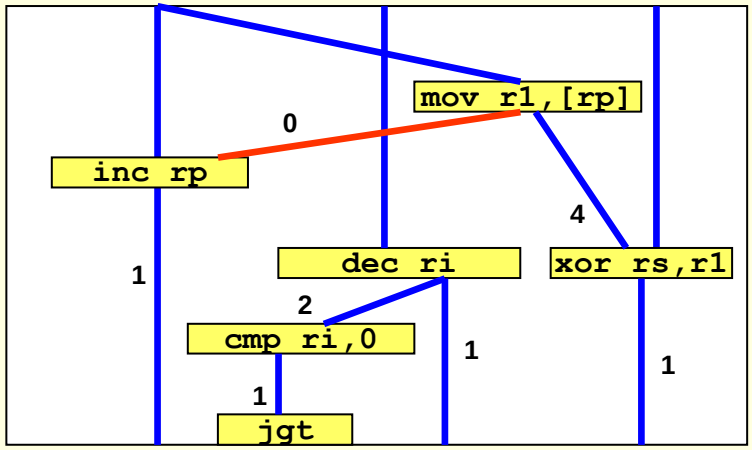
inc r1 dec r1 xor r1,r2 cmp r1,r2	R	MEM	ALU	W
0	1			
1			1	
2				1

mov r1,[r2]	R	MEM	ALU	W
0	1			
1		1		
2		1		
3		1		
4				1

jgt	R	MEM	ALU	W
0			1	

Kapacita procesoru

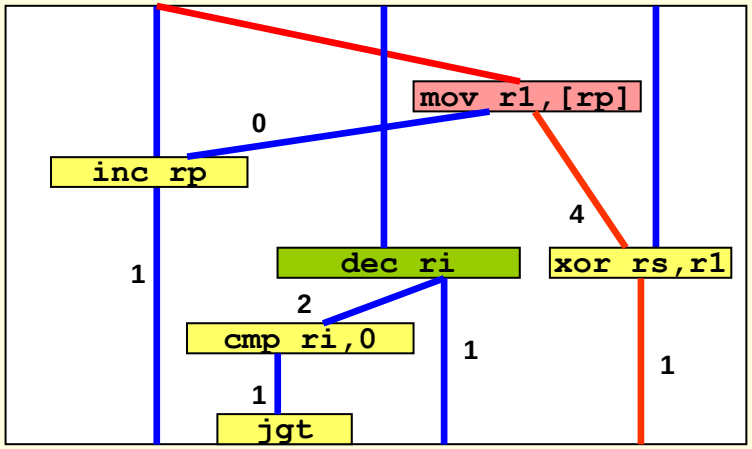
R	MEM	ALU	W
1	1	2	1



Příklad – scheduling

- **Krok 1**
  - Připravené instrukce
    - dec ri
    - mov r1,[rp]
  - Kritická cesta
    - mov r1,[rp] – 5
  - Vybrána instrukce
    - mov r1,[rp]
  - Umístěna do času 0

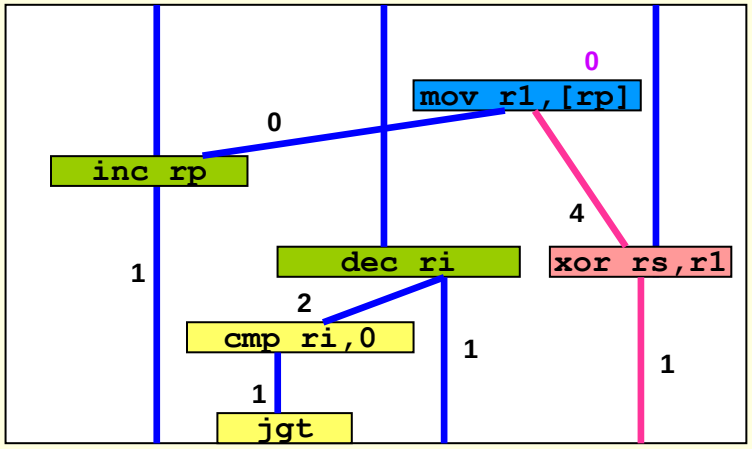
čas	R	MEM	ALU	W
0	mov			
1		mov		
2		mov		
3		mov		
4				mov



Příklad – scheduling

- **Krok 2**
  - Připravené instrukce
    - inc rp
    - dec ri
    - xor rs,r1
  - Kritická cesta
    - xor rs,r1 – 5
  - Vybrána instrukce
    - xor rs,r1
  - Čas 4 určen latencí

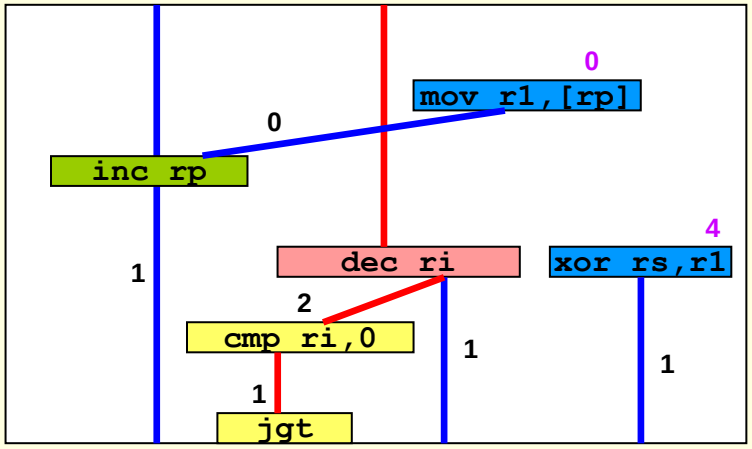
čas	R	MEM	ALU	W
0	mov			
1		mov		
2		mov		
3		mov		
4	xor			mov
5			xor	
6				xor



Příklad – scheduling

- **Krok 3**
  - Připravené instrukce
    - inc rp
    - dec ri
  - Kritická cesta
    - dec ri - 4
  - Vybrána instrukce
    - dec ri
  - Čas 0 vyhovuje latenci
    - Rezervační tabulky jsou obsazeny
  - Zvolen čas 1

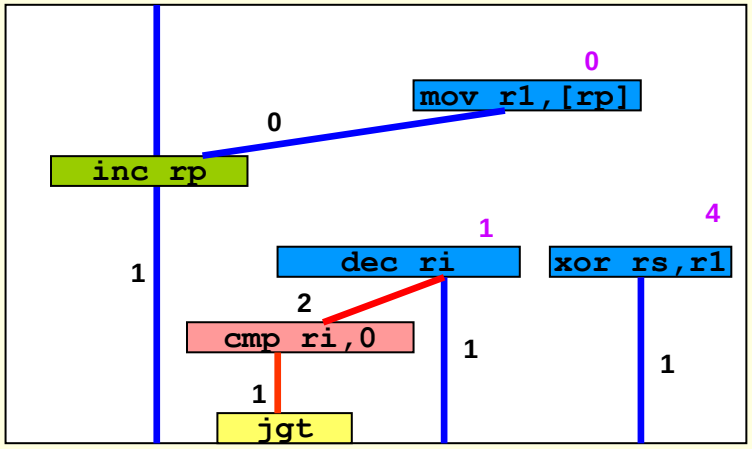
čas	R	MEM	ALU	W
0	mov			
1	dec	mov		
2		mov	dec	
3		mov		dec
4	xor			mov
5			xor	
6				xor



Příklad – scheduling

- **Krok 4**
  - Připravené instrukce
    - inc rp
    - cmp ri,0
  - Kritická cesta
    - cmp ri,0 – 4
  - Vybrána instrukce
    - cmp ri,0 – 4
  - Čas 3 určen latencí

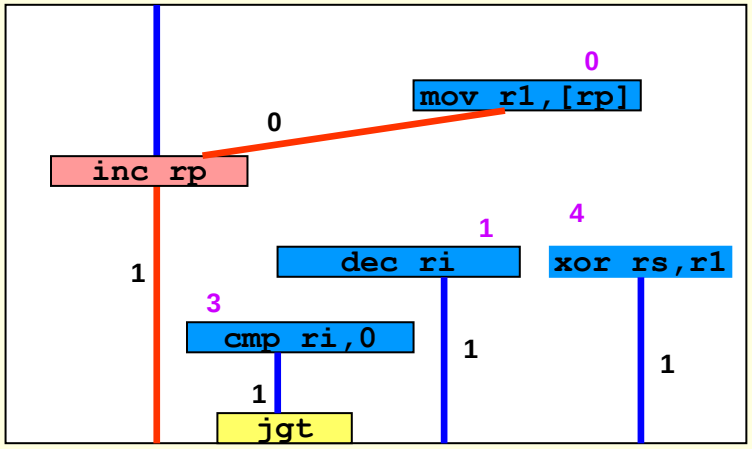
čas	R	MEM	ALU	W
0	mov			
1	dec	mov		
2		mov	dec	
3	cmp	mov		dec
4	xor		cmp	mov
5			xor	cmp
6				xor



Příklad – scheduling

- **Krok 5**
  - Připravené instrukce
    - inc rp
    - (jgt) – musí být poslední
  - Vybrána instrukce
    - inc rp
  - Čas 0 vyhovuje latenci
  - Rezervační tabulky pro časy 0-4 obsazeny
  - Umístěno v čase 5

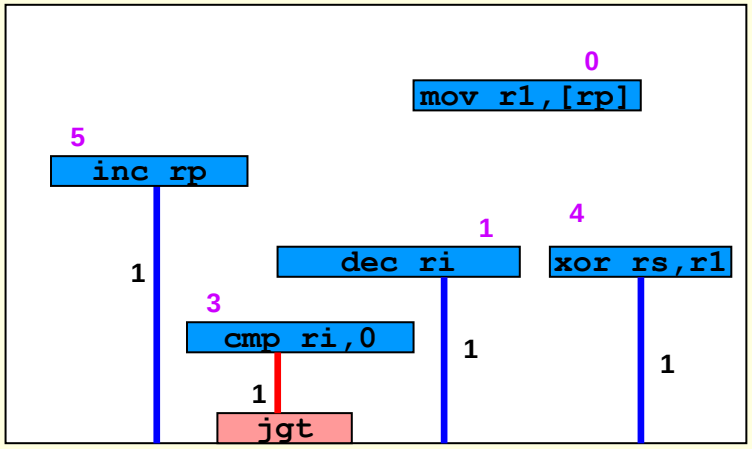
čas	R	MEM	ALU	W
0	mov			
1	dec	mov		
2		mov	dec	
3	cmp	mov		dec
4	xor		cmp	mov
5	inc		xor	cmp
6			inc	xor
7				inc



Příklad – scheduling

- **Krok 6**
  - Připravené instrukce
    - jgt
  - Latenci vyhovuje čas 4
    - Instrukce však musí být poslední
  - Vybrán čas 5

čas	R	MEM	ALU	W
0	mov			
1	dec	mov		
2		mov	dec	
3	cmp	mov		dec
4	xor		cmp	mov
5	inc		xor, jgt	cmp
6			inc	xor
7				inc



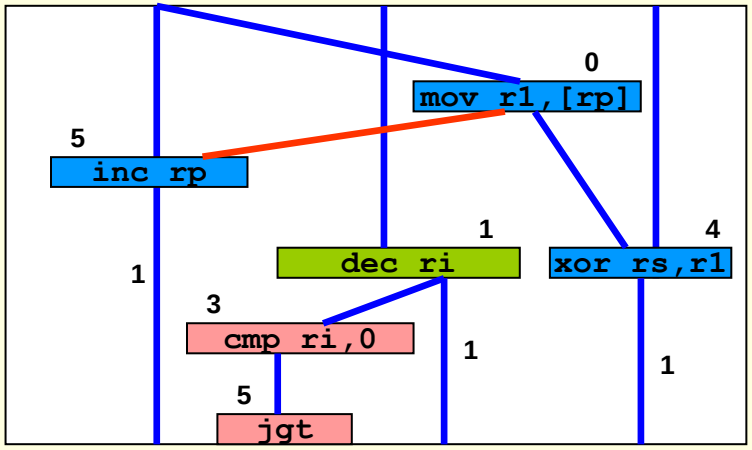
Příklad – scheduling


```
char chksum( char * p, int i)
{
  char s = 0;
  while ( i > 0 )
  {
    s ^= *p++;
    --i;
  }
  return s;
}
```

- Výsledný kód

```
mov r1,[rp]
dec ri
cmp ri,0
xor rs,r1
inc rp
jgt
```

čas	R	MEM	ALU	W
0	mov			
1	dec	mov		
2		mov	dec	
3	cmp	mov		dec
4	xor		cmp	mov
5	inc		xor, jgt	cmp
6			inc	xor
7				inc



- **Základní algoritmus: „List scheduling“**
  - V grafu závislostí s latencemi se pro každý uzel spočte délka kritické, tj. nejdelší cesty ke konci
  - V každém kroku se určí připravené instrukce (a jejich minimální čas) a z nich se vybere nejvhodnější
    - Přednost mají instrukce s největším součtem minimálního času a délky kritické cesty
    - Mezi nimi se vybírá podle dalších heuristik
  - Při správné implementaci má tento postup složitost O(|E|*(log|V| + |R|))
    kde |R| je velikost rezervačních tabulek

- **Vylepšené algoritmy**
  - Posun směrem k exhaustivnímu prohledávání všech možností
  - Branch-and-bound přístup:
    - Zkouší se všechny možnosti v pořadí od nejnadějnější, nalezené list schedulingem
    - Beznadějné pokusy se včas zastaví porovnáváním odhadů úspěšnosti výsledku pomocí kritických cest s doposud známým nejlepším řešením

Příklad – scheduling

Výsledný kód

```
mov r1,[rp]
dec ri
cmp ri,0
inc rp
xor rs,r1
jgt
```

Chování ve smyčce

- Za předpokladu správné predikce skoku procesorem
- Výkon: 1/6 iterace/cyklus
- Využití jednotek:
  - R: 5/6
  - MEM: 3/6
  - ALU: 5/12
  - W: 5/6

čas	R	MEM	ALU	W
0	mov			
1	dec	mov		
2		mov	dec	
3	cmp	mov		dec
4	inc		cmp	mov
5	xor		inc, jgt	cmp
6	mov		xor	inc
7	dec	mov		xor
8		mov	dec	
9	cmp	mov		dec
10	inc		cmp	mov
11	xor		inc, jgt	cmp
12	mov		xor	inc
13	dec	mov		xor
14		mov	dec	
15	cmp	mov		dec

Příklad – scheduling

Efekt kapacity procesoru

R	MEM	ALU	W
2	1	2	2

- Stejné latence i res. tabulky

```
mov r1,[rp]
dec ri
inc rp
cmp ri,0
xor rs,r1
jgt
```

- Výkon: 1/5 iterace/cyklus
- Využití jednotek:
  - R: 5/10
  - MEM: 3/5
  - ALU: 5/10
  - W: 5/10

čas	R	MEM	ALU	W
0	mov, dec			
1	inc	mov	dec	
2	cmp	mov	inc	dec
3		mov	cmp	inc
4	xor		jgt	mov, cmp
5	mov, dec		xor	
6	inc	mov	dec	xor
7	cmp	mov	inc	dec
8		mov	cmp	inc
9	xor		jgt	mov, cmp
10	mov, dec		xor	
11	inc	mov	dec	xor
12	cmp	mov	inc	dec
13		mov	cmp	inc
14	xor		jgt	mov, cmp