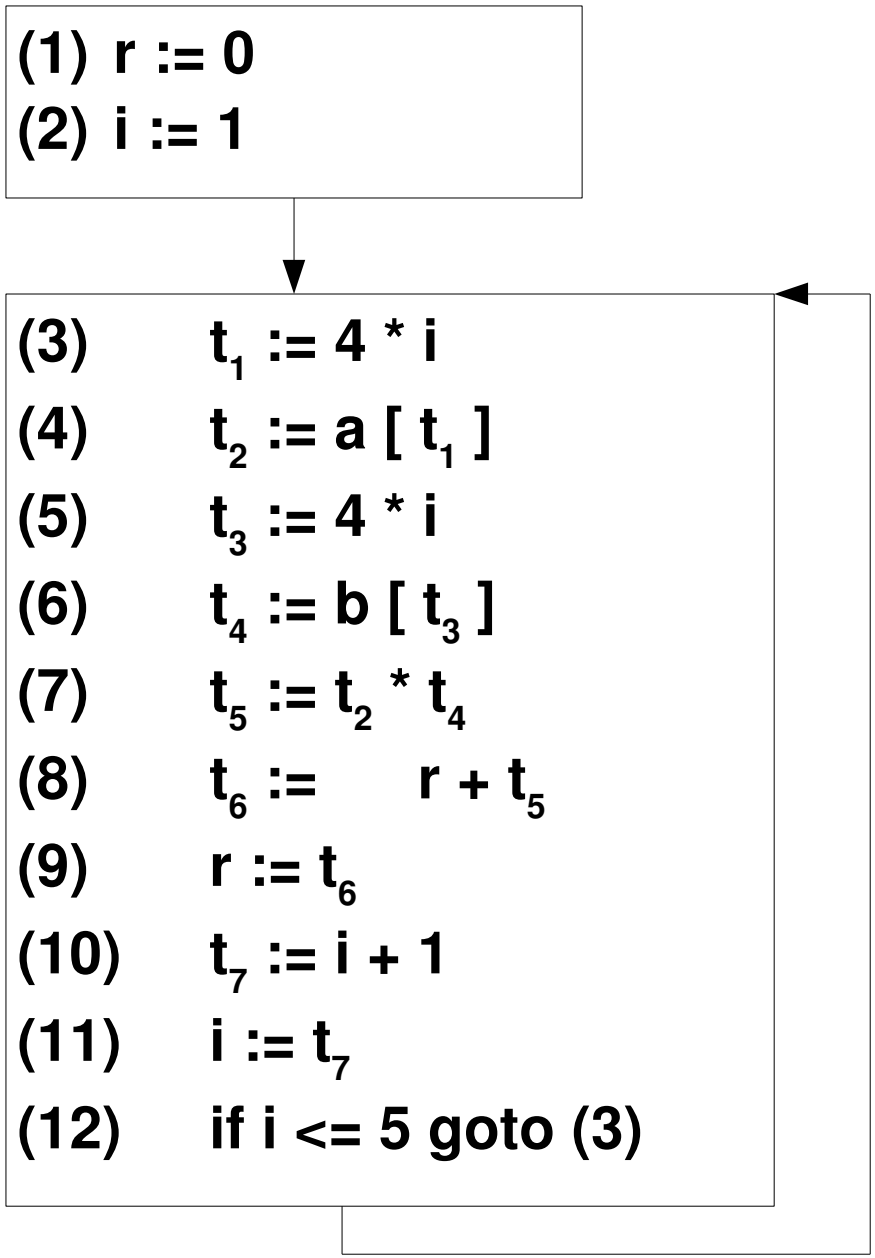
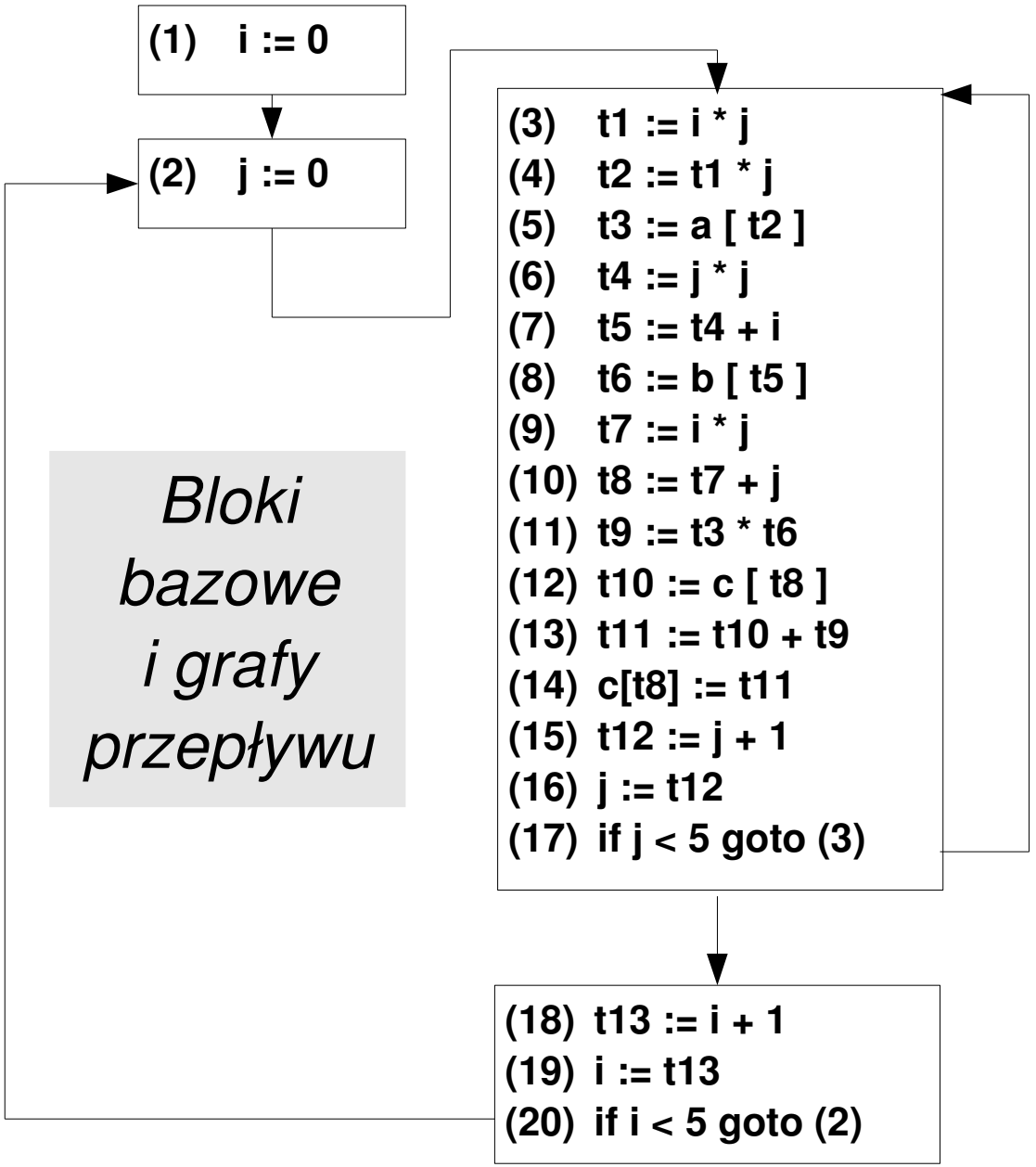
Generacja kodu docelowego

- Zagadnienia związane z generacją kodu
  - Język wejściowy i wynikowy
  - Zarządzanie pamięcią (adresacja)
  - Wybór rozkazów maszynowych (koszty rozkazów)
  - Przydział i wyznaczanie rejestrów
  - Kolejność obliczeń
- Bloki bazowe
  - lider - pierwsza instrukcja, instrukcja bezpośrednio po skoku, cel skoku
  - przekształcenia bloków bazowych
    - zachowujące strukturę:
      - usuwanie wspólnych podwyrażeń (common subexpression),
      - usuwanie martwego kodu,
      - zamiana nazw zmiennych tymczasowych,
      - wymiana dwu niezależnych instrukcji sąsiednich
  - algebraiczne





Bloki bazowe i grafy przepływu

Bloki bazowe c.d

- Określanie ostatniego użycia zmiennej
  - każda zmienna opisywana jest przez parę [status(żywa/martwa), linia]
  - przeglądamy instrukcje od końca (zmienne nietymczasowe są żywe przy wyjściu z bloku)
  - w każdej linii ***i*** postaci ***x := y op z*** zapisujemy aktualny status oraz zapisujemy: x[martwa,-], y[żywa,i], z[żywa,i] w tablicy symboli
- Zmienne tymczasowe a informacja o używaniu zmiennych
- Directed Acyclic Graph - reprezentacja bloku
  - Liście etykietowane nazwami zmiennych lub stałych (z indeksem 0)
  - Wierzchołki wewnętrzne posiadają: etykietę operatora, sekwencję identyfikatorów (aktualne wartości nazw)
  - Przydatne do usuwania wspólnych podwyrażeń

Przykładowy kod

- Rejestry (R0,R1) , pamięć dla zmiennych,
- Instrukcje: MOV, ADD,SUB,MULT (źródło, cel)

$$t_1 := a + b$$
$$t_2 := c + d$$
$$t_3 := e - t_2$$
$$t_4 := t_1 - t_3$$

```
MOV   a, R0
ADD   b, R0
MOV   c, R1
ADD   d, R1
MOV   R0, t1
MOV   e, R0
SUB   R1,R0
MOV   t1,R1
SUB   R0,R1
MOV   R1,t4
```

Zmiana kolejności instrukcji
może polepszyć kod wynikowy

Directed Acyclic Graph

- W(**x**) - wskaźnik na ostatnio utworzony wierzchołek związany z **x**
- Dla każdej instrukcji postaci: (1) **x := y op z** lub (2) **x := op y** lub (3) **x:=y**
  - jeżeli W(y)== undef (podobnie dla z w przypadku (1))
    - W(y) = nowy wierzchołek o etykiecie y
  - w sytuacji:
    - (1) sprawdź, czy istnieje wierzchołek o etykiecie op ,którego lewym dzieckiem jest W(y), a prawym W(z).
      W (2) podobnie. Jeżeli istnieje taki wierzchołek to oznacz go *n* , jeśli nie utwórz nowy.
      W przypadku (3) *n*=W(y)
  - Usuń x z listy identyfikatorów wierzchołka W(x) i dodaj x do listy n.
  - W(x) = *n*

Heurystyczne porządkowanie DAG oraz etykietowanie

```
WW – zbiór niewypisanych wierzchołków wewnętrznych
dopóki WW nie jest pusty:  {
      n – niewypisany wierzchołek z wypisanymi rodzicami
      wypisz n
      dopóki skrajne lewe dziecko (m) dla n nie ma niewypisanych
      rodziców i nie jest liściem: {
                  wypisz m
                  n := m   }
      }
```

Jeżeli n jest **liściem**:

- lewostronnym – **etykieta = 1**
- innym – **etykieta = 0**

w przeciwnym wypadku:

- $n_1, n_2, n_3 \ldots n_i$ dzieci uporządkowane wg etykiet
- etykieta = $\mathbf{max_i\ (etykieta(n_i)+i-1)}$

Przykład generacji kodu

```
gen(n):
   (0) if: n liść reprezentujący zmienną i lewostronne dziecko
          "MOV n, top(rejestry)"
   elseif: n wierzchołek wewnętrzny operatora OP (n1,n2 - dzieci)
      (1) if: et(n2) == 0
             gen(n1)
             "OP n2, top(rejestry)"
      (2) elseif: 1 <= et(n1) < et(n2) oraz et(n1) < rozmiar(rejestry)
             swap(rejestry);  gen(n2); R = pop(rejestry);
             gen(n1);   "OP R,top(rejestry)" ; push(rejestry,R)
             swap(rejestry)
      (3)elseif: 1 <= et(n2) <= et(n1) oraz et(n2) < rozmiar(rejestry)
             gen(n1); R = pop(rejestry); gen(n2);
             "OP top(rejestry), R"
             push(rejestry,R)
      (4) else:
             gen(n2); T = pop(temp); "MOV top(rejestry), T";
             gen(n1); push(temp,T); "OP T, top(rejestry)"
```