ARCHITEKTURA KOMPUTERÓW – proste algorytmy (x86/Pentium – składnia Intel/Borland)

Spis programów

IA-32 – główne różnice składni asemblera Intel / Borland [MS DOS / Windows] oraz AT&T [UNIX / Linux]

(UWAGA: jeden operand musi być rejestrem, chyba że src jest stałą)

IA-32	Intel/Borland [Windows]	AT&T [UNIX / Linux]
lista argumentów w zapisie działania "op" dst:= dst (op) src	op dst, src	op**l** src, dst op**b** src(b), dst(b)
nazwa rejestru	(32b) eax, ebx, … (16b) ax, bx, … (8b) al, ah, bl, bh	(32b) %eax, %ebx, … (16b) —– (brak) (8b) %al, %ah, %bl, %bh,…
stałe: dziesiętne szesnastkowe binarne symboliczne (CONST, STALA,)	1654, –27531, 1011d 1234h, 0ABCh, 101h 0100b, 101111b SCONST, STALA	1654, –27531, 1011, 0d–1012 0x1234, 0xABC, 0x101, 0x–ABC 0b0100, 0b101111 $CONST, $STALA
atrybuty zmiennej: nr segmentu offset (adres) wartość typ (jedn. liczba bajtów) rozmiar	seg ZMIENNA offset ZMIENNA ZMIENNA (w deklaracji: DB, DW, DD,…) size ZMIENNA (w jednostkach rozmiaru)	(brak, model liniowy pamięci) $ZMIENNA ZMIENNA (w deklaracji: .byte, .long, .ascii) (SIZE_ZM =**.** – ZMIENNA) (w bajtach, **.** – wskaźnik lokacji, wyrażenie musi być tuż po deklaracji
wartość elementu zmiennej ideksowanej spod wskazanego adresu (zapis trybu adresowania)	ZMIENNA[ebx, edx*4] [ebx, edx*4] [edx*4], ZMIENNA[edx*4] [ebx], ZMIENNA[ebx], ZMIENNA+5	ZMIENNA(%ebx,%edi,4) (%ebx,%edi,4) (,%edi,4), ZMIENNA(,%edi,4) (%edi), ZMIENNA ZMIENNA+5

ARCHITEKTURA KOMPUTERÓW – proste algorytmy (x86/Pentium – składnia Intel/Borland)

1. Obliczenie wartości bezwzględnej 64-bitowej liczby całkowitej spod adresu ZMIENNA (dwie wersje).

(etykieta)	rozkaz	komentarz
(copy:)	mov eax, ZMIENNA	; przekopiowanie, jeśli wynik ma być w innej zmiennej
	mov ABS-ZM, eax	
	mov eax, ZMIENNA+4	
	mov ABS-ZM+4, eax	; poniżej dwa warianty algorytmu
	or eax, eax	; czy ZMIENNA jest dodatnia?
	jns wynik	
a)	xor eax, eax	; zero (przygotowanie działania 0 – ZMIENNA)
	sub eax, ABS-ZM	; odejmowanie i zapis ABS-ZM = 0 – ZMIENNA
	mov ABS-ZM, eax	
	mov eax, 0	
	sbb eax, ABS-ZM +4	
	mov ABS-ZM+4, eax	; wynik w ABS-ZM
albo b)	not ABS-ZM	; negowanie niższych bitów
	not ABS-ZM +4	; negowanie wyższych bitów
	mov eax, 1	;
	add ABS-ZM, eax	; dodanie "1"
	adc ABS-ZM+4, eax	; wynik w ABS-ZM
wynik:	(into)	; niewykonalne jeśli ZMIENNA = 8000 0000 0000 0000

2. (2A) Obliczanie NWP 32-b liczb naturalnych ZA i ZB wg wzoru Euklidesa (NWP(a,b)=NWP(b, a mod b)).

(etykieta)	rozkaz	komentarz
	mov ebx, ZB	; ecx:ebx= mniejsza =ZBB (jeśli nie, będą przestawione)
	mov eax, ZA	; edx:eax= większa =ZAA
licz:	mov edx, 0	; dzielna (a) w eax, dzielnik (b) w ebx
	div ebx	; reszta (a mod b) w edx, zbędny iloraz w eax
	xchg ebx, eax	; teraz dzielnik (b) będzie dzielną w eax
	xchg ebx, edx	; reszta (a mod b) będzie dzielnikiem w ebx, … zbędny iloraz w edx
	and ebx, ebx	; czy reszta=0, jeśli tak, to NWP jest w eax, reszta jest teraz w ebx
	jnz licz	; licz dopóki reszta ≠0, jeśli =0, to wynik: NWP(a,b) jest w eax

3. Szacowanie √ z 64-b liczby naturalnej spod adresu ZMIENNA wg.zależności 1+3+5+…+(2n–1) = n2

(etykieta)	rozkaz	komentarz
	mov eax, –1	(...(((ZMIENNA–1)|[?<0] –3) |[?<0] –5)|[?<0] –...) – (2ND–1)|[!<0]
	mov edx, –1	; ND = (edx:eax=) –1 – kolejna nieparzysta
	mov ecx, –1	; SQRT = –1 (ZMIENNA<2^{64}→ SQRT<2^{32})
cykl:	add eax, 2	; ND = ND+2 (następna nieparzysta)
	adc edx, 0	
	inc ecx	; SQRT = SQRT+1 (liczba składników sumy nieparzystych)
	sub ZMIENNA, eax	
	sbb ZMIENNA+4, edx	
9	jge cykl	; wynik SQRT (ZMIENNA) w ecx, korekta zbędna

4. Obliczenie wariacji $V(n,k)=n!/(n-k)!$ (silnia , gdy k=1), n – w ecx, k – w esi, $V(n,k)$ w eax

(etykieta)	rozkaz	komentarz
	mov eax, ecx	; n w ecx, k w esi, kopia ecx do eax
	and eax, eax	;
	jz end	; sprawdzenie, czy n=0, wtedy $V(0,k) =^{df} 0$
	sub esi, ecx	; (k–n) do esi
	neg esi	; n–k w esi [–(k–n)]
	mov eax, 1	; początkowy iloczyn (i=n)
licz:	mul ecx	; mnożenie przez bieżący czynnik i
	dec ecx	; kolejny czynnik (i–1)
	cmp ecx, esi	; czy ostatni mnożnik
	jgt licz	; jeśli i>n–k licz dalej, jeśli i=n–k zakończ liczenie
(check:)	and edx, edx	; sprawdź, czy iloczyn jest 32-b (w eax – edx:eax = 0:eax)
	jnz error	; przekroczony zakres, wynik >2^{32}
end:		;

ARCHITEKTURA KOMPUTERÓW – proste algorytmy (x86/Pentium – składnia Intel/Borland)

5. Obliczenie kombinacji C(n,k) = n!/[k!(n–k)!], : n – liczba z rejestru ecx, k – liczba z rejestru esi, wynik w eax
algorytm: C(n,k) = {[...{[(n*(n–1)/2]*(n–2)/3}*(n–3)/4 ....]*(n–k+1)/(k–1)}*(n–k+1)/k

(etykieta)	rozkaz	komentarz
(prep:)	mov ebx, esi	; argument *k* do ebx
	add ebx, ebx	; oblicz 2*k*
	sub ebx, ecx	; oblicz 2*k*–*n*
	jle hop	; jeśli *k*≤*n*/2 przeskocz
	sub esi, ebx	; zastąp *k* przez *k*–(2*k*–*n*) = *n*–*k*
	mov eax, 0	; *C*(*n*,*k*) = 0 gdy *k*>*n*
	jlt end	; koniec gdy *k*>*n*
hop:	mov eax, 1	; *C*(*n*,*k*) = 1 gdy *k*=0 lub *n*–*k*=0
	and esi, esi	
	jz end	
	mov edx, 0	; *n* jest w ecx, *k* lub *n*–*k* jest w esi
	mov ebx, 1	;
	mov eax, ecx	; *n* do eax
comb:	dec ecx	; kolejny czynnik (*n*–*i*) iloczynu
	mul ecx	;
	inc ebx	; kolejny dzielnik *i*+1
	div ebx	; *C*(*n*,*i*+1) = *C*(*n*,*i*)*(*n*–*i*))/(*i*+1)
(check:)	and edx, edx	; sprawdź, czy iloczyn jest w eax (edx:eax = 0:eax)
	jnz error	; przekroczony zakres, wynik >2^{32}
	cmp ebx, esi	; jeśli *i*+1<*k*, kontynuuj
	jlt comb	
end:		; wynik w eax

6. Wyszukiwanie największej całkowitej z liczb 64-bitowych umieszczonych w tablicy LICZBY (*big endian*):

(etykieta)	rozkaz	komentarz
	mov edx, [LICZBY]	; edx = max-liczba$_H$
	mov eax, [LICZBY+4]	; eax = max-liczba$_L$
	mov esi, 2	
	mov ecx, ILE–1	
	jcxz koniec	
pocz:	inc esi	
	mov ebx, [LICZBY+4*esi]	; ebx = kolejna$_H$
	mov ebp, [LICZBY+4+4*esi]	; ecx = kolejna$_L$
	cmp edx,ebx	
	jgt cykl	; powtórz gdy max-liczba > kolejna
	jne dalej	; czy kolejna jest wieksza?
	cmp eax,ebp	
	jge cykl	; kolejna nie większa od max-liczby
dalej:	mov edx, ebx	; kolejna jest wieksza od max-liczby, więc przestaw
	mov eax, ebp	
cykl:	inc esi	;
	loop pocz	; powtórz jeśli nie przeszukano całego zbioru, max w edx:eax

7. Obliczenie *N*-tej liczby Fibonacciego

(etykieta)	rozkaz	komentarz
	mov ecx, N	; N – liczba naturalna (>0)
	mov eax, 0	; wartość początkowa F(0)=0
	mov ebx, 1	; F(1)=1
	cmp ebx, N	; czy N=1
	je out	
fibonac:	add eax, ebx	; akumulacja: F(i–1) w eax, F(i) w ebx, F(i+1) do eax
	xchg ebx, eax	; zamień F(i+1) z F(i)
	loop fibonac	; zmniejsz licznik i sprawdź, czy ostatnia
out:		; F(N) w ebx zaś F(N–1) w eax

ARCHITEKTURA KOMPUTERÓW – proste algorytmy (x86/Pentium – składnia Intel/Borland)

8. Szyfr Cezara: tekst w buforze TEKST, ostatni znak NULL (kod 0), klucz w cl, szyfrogram w buforze CEZAR

(etykieta)	rozkaz	komentarz
	mov esi, offset TEKST	; usuwanie spacji (0x20 = ' ' = SP)
	mov edi, offset CEZAR	;
pocz:	mov al, [esi]	;
	inc esi	;
	cmp al, 0	;
	jeq end	;
	or al, 20h	; wszystkie cyfry małe (maska 0010 0000)
	cmp al, ' '	; usuwanie spacji
	jeq pocz	;
	add al, cl	;
	cmp al, 'z'	;
	jle omijaj	;
	sub al, 1Ah	; (1Ah=26 liter w alfabecie)
omijaj:	mov [edi], al	;
	inc edi	;
	jmp pocz	;
end:	mov [edi], al	; znak końca (NULL) do bufora szyfrogramu

9. Obliczenie wartości liczby wprowadzanej jako sekwencja cyfr ASCII (z bufora LICZBA)

(etykieta)	rozkaz	komentarz
	mov ebx, 10	; max 9 cyfr, SIZE – rozmiar liczby, max. 9 cyfr
	mov eax, 0	;
	mov esi, 0	;
pocz:	mov cl, LICZBA [esi]	;
	and ecx, 0Fh	;
	add eax, ecx	;
	mul ebx	;
	inc esi	;
	cmp esi, SIZE	;
	jlt pocz	;

10. Mnożenie 64-b liczb naturalnych (opcja: całkowitych w U2) danych pod adresami MNOŻNA i MNOŻNIK.

(etykieta)	rozkaz	komentarz
		; całkowite: $A\cdot M=(A_+-2^{63})\cdot(M_+-2^{63})= A_+\cdot M_+ - 2^{63}\,(A_++M_+) -2^{126}$
	mov eax, MNOŻNA	; $A\cdot M = A_L\cdot M_L+2^{32}(A_H\cdot M_L+A_L\cdot M_H)+ 2^{64}(A_H\cdot M_H)$
	mul MNOŻNIK	; edx:eax $=(A_L\cdot M_L)_H:(A_L\cdot M_L)_L$
	mov ILOCZYN, eax	; $ILOCZYN_{LL}$ = eax = $(A_L\cdot M_L)_L$ (gotowe $(A\cdot M)_{LL}$)
	mov ebx, edx	; ebx: = $(A_L\cdot M_L)_H$ (do dalszego przetwarzania)
	mov eax, MNOŻNA+4	
	mul MNOŻNIK	; edx:eax $=(A_H\cdot M_L)_H:(A_H\cdot M_L)_L$
	add ebx, eax	; ebx: = $[(A_L\cdot M_L)_H+(A_H\cdot M_L)_L] \bmod 2^{32}$
	adc edx, 0	; (bez przekroczenia zakresu, bo iloczyn $\le(2^{32}-1)^2=2^{64}-2^{33}+1$)
	mov ecx, edx	; ecx = skorygowany $(A_H\cdot M_L)_H$
	mov eax, MNOŻNA	
	mul MNOŻNIK+4	; edx:eax $=(A_L\cdot M_H)_H:(A_L\cdot M_H)_L$
	add ebx, eax	; ebx: = $[[(A_L\cdot M_L)_H+(A_H\cdot M_L)_L] \bmod 2^{32} +(A_L\cdot M_H)_L] \bmod 2^{32}$
	adc ecx, edx	; ecx = [skorygowany $(A_H\cdot M_L)_H$ + $(A_L\cdot M_H)_H$+ CY] mod 2^{32}
	mov ILOCZYN+4, ebx	; $ILOCZYN_{LH}$=ebx=$(A_L\cdot M_L)_H+(A_H\cdot M_L)_L+(A_L\cdot M_H)_L$ (gotowe $(A\cdot M)_{LH}$)
	mov ebx, 0	
	adc ebx, 0	; przeniesienie na $ILOCZYN_{HL}$
	mov eax, MNOŻNA+4	
	mul MNOŻNIK+4	; edx:eax $=(A_h\cdot M_H)_H:(A_H\cdot M_H)_L$
	add eax, ecx	
	adc adx, ebx	
	mov ILOCZYN+8, eax	; $ILOCZYN_{HL}$
23	mov ILOCZYN+12, edx	; $ILOCZYN_{HH}$

ARCHITEKTURA KOMPUTERÓW – proste algorytmy (x86/Pentium – składnia Intel/Borland)

11. Generowanie liczb pierwszych $<2^{32}$ wg.algorytmu „sito Eratostenesa" (usuwanie z listy wielokrotności liczby *p*).

(etykieta)	rozkaz	komentarz
	cld	; *k*–zakres, HK=*k*/2, SQK=sqrt(*k*), ERAT– tablica
	mov eax, –1	; edx – indeks wyszukiwania, eax – indeks bieżący, esi – nr liczby
	mov edi, offset ERAT	
	mov ecx, HK	
	rep stosd	; wypełnienie tablicy ERAT stałą „–1" (stała może być dowolna)
	mov esi, 0	; esi – indeks liczby pierwszej
	mov edx, 1	; edx – indeks bieżący
pocz:	mov ebx, edx	
	add ebx, ebx	
	add ebx, 1	; ebx = 2*edx+1
	cmp edx, SQK	
	jgt hop	
	mov eax, edx	
pet:	add eax, ebx	; indeksy komórek z wielokrotnościami l.p.
	mov [ERAT +4*eax], 0	; zerowanie
	cmp eax, HK	
	jlt pet	
hop:	inc esi	
	mov [ERAT +4*esi], ebx	; kolejna liczba pierwsza do kolejnej komórki ERAT (lub innej tab)
cykl:	inc edx	
	cmp [ERAT +4*edx], 0	
	jz cykl	
	cmp edx, HK	
	jlt pocz:	
25	mov ERAT, 2	

12. Mnożenie 64-b liczb naturalnych (opcja: całkowitych w U2) danych pod adresami MNOŻNA i MNOŻNIK.

(etykieta)	rozkaz	komentarz
		; całkowite: $A\cdot M=(A_+-2^{63})\cdot(M_+-2^{63})= A_+\cdot M_+ - 2^{63}(A_++M_+) -2^{126}$
	mov eax, MNOŻNA	; $A\cdot M = A_L\cdot M_L+2^{32}(A_H\cdot M_L+A_L\cdot M_H)+ 2^{64}(A_H\cdot M_H)$
	mul MNOŻNIK	; edx:eax $=(A_L\cdot M_L)_H:(A_L\cdot M_L)_L$
	mov ILOCZYN, eax	; $ILOCZYN_{LL}$ = eax = $(A_L\cdot M_L)_L$ (gotowe $(A\cdot M)_{LL}$)
	mov ebx, edx	; ebx: = $(A_L\cdot M_L)_H$ (do dalszego przetwarzania)
	mov eax, MNOŻNA+4	
	mul MNOŻNIK	; edx:eax $=(A_H\cdot M_L)_H:(A_H\cdot M_L)_L$
	add ebx, eax	; ebx: = $[(A_L\cdot M_L)_H+(A_H\cdot M_L)_L] \bmod 2^{32}$
	adc edx, 0	; (bez przekroczenia zakresu, bo iloczyn $\le(2^{32}-1)^2=2^{64}-2^{33}+1$)
	mov ecx, edx	; ecx = skorygowany $(A_H\cdot M_L)_H$
	mov eax, MNOŻNA	
	mul MNOŻNIK+4	; edx:eax $=(A_L\cdot M_H)_H:(A_L\cdot M_H)_L$
	add ebx, eax	; ebx: = $[[(A_L\cdot M_L)_H+(A_H\cdot M_L)_L] \bmod 2^{32} +(A_L\cdot M_H)_L] \bmod 2^{32}$
	adc ecx, edx	; ecx = [skorygowany $(A_H\cdot M_L)_H$ + $(A_L\cdot M_H)_H$+ CY] mod 2^{32}
	mov ILOCZYN+4, ebx	; $ILOCZYN_{LH}$=ebx=$(A_L\cdot M_L)_H+(A_H\cdot M_L)_L+(A_L\cdot M_H)_L$ (gotowe $(A\cdot M)_{LH}$)
	mov ebx, 0	
	adc ebx, 0	; przeniesienie na $ILOCZYN_{HL}$
	mov eax, MNOŻNA+4	
	mul MNOŻNIK+4	; edx:eax $=(A_h\cdot M_H)_H:(A_H\cdot M_H)_L$
	add eax, ecx	
	adc adx, ebx	
	mov ILOCZYN+8, eax	; $ILOCZYN_{HL}$
23	mov ILOCZYN+12, edx	; $ILOCZYN_{HH}$