Laboratorium programowania niskopoziomowego

LAB 7 – Operacje na liczbach BCD.

BCD WPROWADZENIE

BCD (binary coded decimal) jest zapisem cyfr dziesiętnych w kodzie binarnym. Jeden bajt może dzielić się na 2 części po 4 bity, z których każda może przyjąć wartość 0-9. Cały bajt reprezentuje zatem liczby od 0 do 99, w których cyfra jedności jest zapisana w młodszych 4 bitach, a cyfra dziesiątek w starszych 4 bitach. Taka forma przechowywania liczb BDC nazywa się spakowanym BCD. Natomiast niespakowane BCD oznacza, że najmłodsze 4 bity przechowują cyfrę BCD, a starsze nie biorą udziału w obliczeniach. Uwaga – dla niespakowanych liczb BCD starsze 4 bity należy wyzerować, aby podczas procedury arytmetycznej prawidłowo działały przeniesienie i pożyczka!

Liczba dziesiętna **1234** zapisana w BCD nie spakowanym:

```
char *bcd = new char[4];
bcd[0] = 0b 0000 0001; // 1
bcd[1] = 0b 0000 0010; // 2
bcd[2] = 0b 0000 0011; // 3
bcd[3] = 0b 0000 0100; // 4
```

Liczba dziesiętna **1234** zapisana w BCD spakowanym:

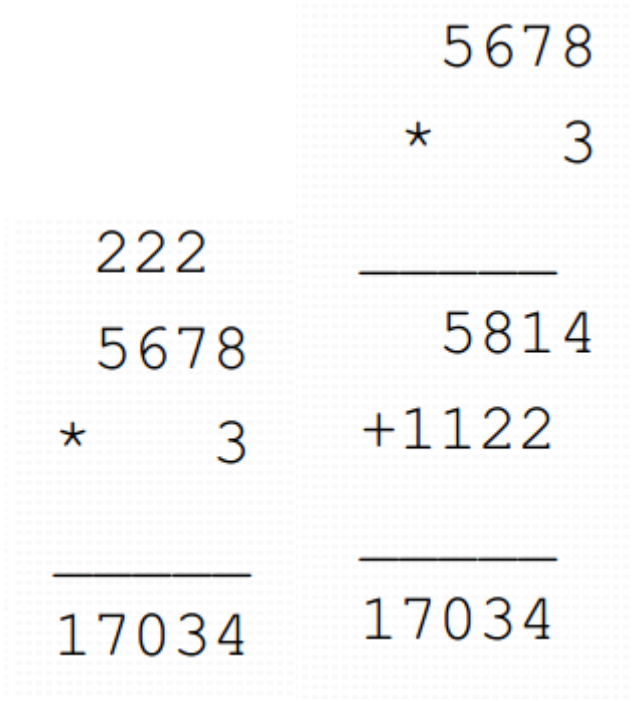
```
char *bcd = new char[2];
bcd[0] = 0b 0001 0010b; // 1 2
bcd[1] = 0b 0011 0100b; // 3 4
```

Arytmetyka BCD opiera się o klasyczne operacje arytmetyczne, obliczane „pod kreskę”. Poniżej przedstawiono schemat dodawania, dla odejmowania jest on analogiczny.

```
  286
+ 111
-----
  397
```

```
    1
  286
+ 321
-----
  607
```

Poniżej przedstawiono schemat mnożenia liczb, który wymaga więcej operacji niż w przypadku dodawania:



Kod dodawania liczb BCD w asemblerze x86:

```
void addBDC(char* numberIN1, char* numberIN2, char* numberOUT, int Size)
{
	__asm {
	push ebx;
	mov ecx, Size;
	mov esi, numberIN1;
	mov edi, numberIN2;
	mov ebx, numberOUT;

	clc;                            //CF = 0
petla: mov   al, [esi+ecx-1];       //pobierz cyfrę źródła1
	adc   al, [edi+ecx-1];          //dodaj cyfrę źródła2 z przeniesieniem
	aaa;                            //korekta po dodawaniu
	mov   [ebx+ecx-1], al;          //zapamiętaj cyfrę
	dec   ecx;
	jnz   petla;                    //CF nie zmieniło się od AAA!!!
	pop   ebx;
	}
}
```

Kod mnożenia liczb BCD:

```
void mulBCD(char*a, char*b, int n, int m, char*w) {
	__asm {
			pushad;
			pushf;
			mov ecx, m		//ilość cyfr mnożnika - m
			mov edi, b		//adres mnożnika
			add edi, ecx
			mov edx, w		//adres wyniku – WYZEROWAĆ WYNIK PRZED WYWOŁANIEM!
			add edx, ecx
		po_m:
			push ecx		//zabezpiecz m ilość pozostałych cyfr mnożnika
			dec edx			//cyfra wyniku od m-1 do 0
			dec edi			//cyfra mnożnika od m-1 do 0
			mov bl, [edi]		//cyfra mnożnika do bl
			mov ecx, n		//ilość cyfr mnożnej - n
			mov esi, a		//adres mnożnej
		iloczyn:
			mov al, [esi]		//cyfra mnożnej - od najstarszej
			mul bl			//razy cyfra mnożnika
			aam			//korekta po mnożeniu
			push ax			//ah - przeniesienie z mnożenia, al - wynik na stos
			inc esi
			loop iloczyn
			mov ecx, n
			pop ax			//najmłodsze ze stosu
			add al, [edx + ecx] //i dodajemy cyfrę do wyniku
			aaa			//korekta po dodawaniu
			mov [edx + ecx], al
			dec ecx
			jz one_n		//jeśli tylko jednocyfrowa mnożna
		SumaBCD:
			mov bl, ah		//przeniesienie do bl
			pop ax			//następna cyfra i przeniesienie ze stosu
			add al, [edx + ecx] //dodajemy kolejną cyfrę wyniku
			aaa
			add al, bl		//plus poprzednie przeniesienie
			aaa
			mov[edx + ecx], al	//i do wyniku
			loop SumaBCD
		one_n:
			mov [edx], ah		//ostatnie przeniesienie do wyniku
			pop ecx			//ilość pozostałych cyfr mnożnika
			loop po_m
			popf
			popad
	}
}
```

Bardzo istotne jest właściwe przygotowanie kodu w C++, należy odpowiednio przygotować tablice, które użytkownik uzupełni konkretnymi wartościami.

```
int main()
{
	int maxSize = 30;

	int actualSize1 = 0;
	int actualSize2 = 0;
	int actualSizeOUT = 0;
	int actualSizeOUT_ = 0;
```

```
char* numberIN1 = new char[maxSize];
char* numberIN2 = new char[maxSize];
char* numberIN1_ = new char[maxSize];
char* numberIN2_ = new char[maxSize];
char* numberOUT = new char[maxSize];
char* numberOUT_ = new char[maxSize];
char c;

puts("Podaj liczbe 1:");
do {
	c = (char)getchar();
	numberIN1[actualSize1] = c;
	numberIN1_[actualSize1] = c;
	actualSize1++;
} while (!((c == '\n') || (actualSize1 >= maxSize)));
actualSize1--;
for (int i = maxSize - 1, j = 0; i >= 0; --i, ++j)
{
	if (j < actualSize1)
		numberIN1[i] = numberIN1[actualSize1 - j - 1];
	else
		numberIN1[i] = '0';
}

puts("Podaj liczbe 2:");
do {
	c = (char)getchar();
	numberIN2[actualSize2] = c;
	numberIN2_[actualSize2] = c;
	actualSize2++;
} while (!((c == '\n') || (actualSize2 >= maxSize)));
actualSize2--;
for (int i = maxSize - 1, j = 0; i >= 0; --i, ++j)
{
	if (j < actualSize2)
		numberIN2[i] = numberIN2[actualSize2 - j - 1];
	else
		numberIN2[i] = '0';
}

printf("\nPierwsza liczba do dodawania/odejmowania to: %.*s\n", maxSize, numberIN1);
printf("Druga liczba do dodawania/odejmowania to:    %.*s\n", maxSize, numberIN2);
printf("\nPierwsza liczba do mnozenia to: %.*s\n", actualSize1, numberIN1_);
printf("Druga liczba do mnozenia to:    %.*s\n", actualSize2, numberIN2_);

for (int i = 0; i < maxSize; ++i) numberIN1[i] &= 15;
for (int i = 0; i < maxSize; ++i) numberIN2[i] &= 15;
for (int i = 0; i < maxSize; ++i) numberIN1_[i] &= 15;
for (int i = 0; i < maxSize; ++i) numberIN2_[i] &= 15;
for (int i = 0; i < maxSize; ++i) numberOUT[i] = '0';
for (int i = 0; i < maxSize; ++i) numberOUT_[i] = '0';

addBDC(numberIN1, numberIN2, numberOUT, maxSize);

mulBCD(numberIN1_, numberIN2_, actualSize1, actualSize2, numberOUT_);

for (int i = 0; i < maxSize; ++i) numberOUT[i] |= 48;
for (int i = 0; i < maxSize; ++i) numberOUT_[i] |= 48;
printf("\nWynikowa liczba z dodawania/odejmowania to: %.*s\n", maxSize, numberOUT);
printf("Wynikowa liczba z mnozenia to: %.*s\n", maxSize, numberOUT_);

for (int i = 0; i < maxSize; ++i, actualSizeOUT++) if ((char)numberOUT[i] != '0') break;
```

```
	for (int i = 0; i < maxSize; ++i)
	{
		if (i + actualSizeOUT < maxSize)
			numberOUT[i] = numberOUT[i + actualSizeOUT];
		else
			numberOUT[i] = ' ';
	}
	actualSizeOUT = maxSize - actualSizeOUT;

	actualSizeOUT_ = actualSize1 + actualSize2;
	if (numberOUT_[0] == '0')
	{
		for (int i = 0; i < maxSize; ++i)
		{
			if (i + 1 < maxSize)
				numberOUT_[i] = numberOUT_[i + 1];
			else
				numberOUT_[i] = ' ';
		}
		actualSizeOUT_--;
	}
	printf("\nWynik dodawania/odejmowania to: %.*s", actualSizeOUT, numberOUT);
	printf("\nWynik mnozenia to: %.*s\n\n", actualSizeOUT_, numberOUT_);
	system("PAUSE");
	delete[] numberIN1;
	delete[] numberIN2;
	delete[] numberOUT;
	delete[] numberOUT_;
    return 0;
}
```

Powyższy kod sam dopasowuje ilość cyfr w liczbie – odpowiednio uzupełniając zerami miejsca przed liczbą, oraz sam przetwarza kod ASCI na kod BCD.

Zadania do samodzielnego wykonania:

1. Napisz kod dla odejmowania liczb BCD
2. Sprawdź co się stanie jak wynik wyjdzie ujemny podczas odejmowania.
3. Sprawdź co się stanie jak wynik wyjdzie za duży i nie mieści się w tablicy wynikowej (możesz odpowiednio zmienić tablicę)
4. Przetestuj mnożenie BCD.
5. Napisz dodawanie/odejmowanie/mnożenie liczb zmiennoprzecinkowych np.: 123.45 i 35.54. Uwaga należy zmodyfikować kod C++, aby użyć gotowych podprogramów dla liczb całkowitych i właściwie obsłużyć z kropkę/przecinek.
6. Zastanów się, czy można wykonać powyższe zadania w architekturze X64 – uzasadnij swoją odpowiedź.
7. *Bazując na zdobytej wiedzy spróbuj wykonać samodzielnie dzielenie liczb BCD.