ЛАБОРАТОРНА РОБОТА №2-1

Реалізація основних програмних конструктів мовою Асемблера. Використання асемблерних вставок у програмах мовою Паскаль

Мета роботи – вивчення методів реалізації мовою Асемблера основних виконавчих операторів мови Паскаль, ознайомлення з методикою включення текстів програм мовою Асемблера в програми мовою Паскаль.

2-1.1. Зміст роботи

Робота виконується на двох заняттях. На першому занятті на основі запропонованої програми мовою Паскаль студенти створюють файл, що містить результати трансляції кожного оператора Паскаль програми в інструкції команд Асемблера, та вивчають методи реалізації мовою Асемблера основних операторів мови Паскаль. На другому занятті створюють у Паскаль програмі асемблерну вставку, що оптимізує, якщо можливо, програму за обсягом і/або швидкодією.

2-1.2. Теоретичні відомості

Подання основних даних в ЕОМ та елементарні операції над ними

1) Розміри, типи, внутрішня логічна структура даних

Дані, які зберігаються в пам'яті і оброблюються командами, відрізняються своїм розміром (кількістю комірок пам'яті або байтів, які вони займають) та логічною структурою. Так, наприклад, компілятор TP мови Паскаль для типу integer відводить 2 байти, а Delphi та C++ – 4 байти для подання цілих зі знаком у доповняльному коді. Асемблер має синтаксичні правила, які дозволяють однозначно визначати розмір даних у байтах, з якими оперує окремо взята команда. У цьому є певна аналогія із завданням типів даних у мовах високого рівня. Саме тому в Асемблері *розмір даних в байтах*

називають типом даних, ігноруючи (на відміну від мов високого рівня) *внутрішню логічну структуру* даних, яка визначається та інтерпретується виключно операціями, що задаються в командах. При цьому в апаратурі ЕОМ і в Асемблері відсутні засоби контролю типів даних, які програмно реалізовані в трансляторах мов високого рівня. Мовою Асемблера задаються, а транслятором контролюються лише розміри даних. Тобто будь-яка команда буде читати з пам'яті стільки байтів, скільки їй потрібно, і буде обробляти їх так, як їй потрібно, незалежно від того, дані якого розміру та якої внутрішньої структури з точки зору програміста там розташовані. Контроль за відповідністю розміру та внутрішньої логічної структури даних командам ЕОМ у програмах мовою Асемблера повністю покладається на програміста. У випадку програм мовами високого рівня цю відповідність забезпечують транслятори, використовуючи описи типів даних, які є обов'язковими.

2) Адресація багатобайтних даних та порядок розташування байтів

В сучасних комп'ютерних системах стандартом фактично є побайтова організація ОЗП, тобто у комірці пам'яті, яка має свій оригінальний номер (фізичну адресу) міститься один байт. Це призвело до деяких ускладнень при організації обробки багатобайтних даних командами процесора.

В процесорах Intel принято, що *молодші байти багатобайтних даних розташовуються в ОЗП за молодшими адресами, а адресою багатобайтних даних є адреса їх молодшого байту.*

Нехай, наприклад, в пам'яті необхідно зберігати число 19498, яке в двійковій системі числення має вигляд 100110000101010, а у шістнадцятковій – 4c2a. Тоді за фізичною адресою Addr буде записано байт 2a (двійкове 00101010), а за фізичною адресою Addr+1 – байт 4c (двійкове 01001100). Фізичною адресою цього числа буде Addr. Як видно з прикладу, десяткова система числення при цьому досить незручна.

3) Цілі числа без знаку

В сучасних інформаційних технологіях цілі числа без знаку можуть мати розмір 1, 2, 4 або 8 байт. Мінімальним значенням цих чисел є 0, а максимальне

обчислюється за формулою 2^{k*8} -1, де k – розмір числа в байтах. Згідно з вищезазначеним, k вибирається із ряду 1,2,4,8. У табл. 2-1.1 надані діапазони значень цілих чисел без знаку в залежності від k.

Таблиця 2-1.1

Діапазони значень цілих чисел без знаку

k	1	2	4	8
Діапазон значень	0..255	0..65535	0..4294967295	0..18446744073709551615

Особливістю апаратної реалізації основних арифметичних операцій чисел без знаку є однакова розрядність операндів. У випадку відмінної розрядності чисел, розрядність числа з меншою розрядністю необхідно збільшити до розрядності числа з більшою розрядністю. Це досягається записом 0 у всі додаткові розряди. Починаючи з процесорів Pentium (відповідно у всіх сучасних процесорах), реалізується універсальна команда *MOVZX* для розширення розрядності беззнакового числа.

При додаванні чисел їх сума може перевищувати максимальне значення. В цьому випадку виникає перенесення із старшого розряду. Для фіксації наявності (або відсутності) перенесення використовується біт переносу ***cf*** (*carry flag*) у регістрі ознак. Вміст розряду переносу визначає, таким чином, наявність переповнення при додаванні беззнакових цілих чисел.

При відніманні беззнакових чисел також формується ознака переносу, яка фактично визначає позику із віртуального старшого розряду, який умовно знаходиться за старшим розрядом двійкового подання числа. Значення ознаки переносу дорівнює 1, якщо зменшуване менше за від'ємник, і 0 в інших випадках.

Таким чином, значення ознаки переносу, сформоване після операції віднімання беззнакових цілих чисел можна використовувати як результат їх порівняння. Тобто, якщо після виконання операції X-Y *cf=1*, то X<Y, а якщо *cf=0*, то X≥Y.

Для ідентифікації порівнянь X=Y, X>Y, та X≤Y необхідно використовувати додаткову ознаку – *ознаку нуля*. Ознака приймає значення 1, якщо результатом є нуль, в усіх інших випадках ознака нуля приймає значення 0. Ознака нуля позначається як ***zf*** (*zero flag*) і також міститься в регістрі ознак.

Команда процесора порівняння (мнемокод *CMP* - *CoMPare*) виконує операцію віднімання операндів, встановлюючи відповідні значення ознак, але результат віднімання нікуди не записує. Отже, після виконання операції X-Y для беззнакових чисел маємо:

- X>Y, якщо ((*cf*=0) і (*zf*=0)),
- X≥Y, якщо *cf*=0,
- X=Y, якщо *zf*=1,
- X≠ Y, якщо *zf*=0,
- X≤Y, якщо ((*cf*=1) і (*zf*=1)),
- X<Y, а якщо *cf*=1.

Сформовані ознаки використовуються в *командах передачі управління за умовою* процесорів 80x86 (Pentium) (табл. 2-1.2).

Таблиця 2-1.2

Команди передачі управління за умовою для операндів без знаку

Співвідношення між операндами	Значення ознак при відніманні (X - Y)	Найменування співвідношення	Мнемокод команди передачі управління за умовою
X > Y	(***cf***=0) і (***zf***=0)	Вище (або не нижче і не дорівнює)	**JA** (або **JNBE**)
X ≥ Y	***cf***=0	Вище чи дорівнює (або не нижче)	**JAE** (або **JNB**, або **JNC**)
X = Y	***zf***=1	Дорівнює	**JE** (або **JZ**)
X ≠ Y	***zf***=0	Не дорівнює	**JNE** (або **JNZ**)
X ≤ Y	***cf***=1 чи ***zf***=1	Нижче чи дорівнює (або не вище)	**JBE** (або **JNA**)
X < Y	***cf***=1	Нижче (або не вище і не дорівнює)	**JB** (або **JNAE**, або **JC**)

Як видно, одна й та сама команда передачі управління за умовою може мати декілька мнемокодів, що надає можливість програмістам краще відтворювати алгоритм програми і, тим самим, покращити його розуміння.

Інтерпретація результатів операції зсуву для беззнакових чисел. Лінійний зсув вліво (команда *SHL – SHift Left*) на один розряд інтерпретується як операція множення на 2. Вміст ознаки перенесення, як і при додаванні, визначає факт переносу. *Лінійний зсув вправо* (команда *SHR – SHift Right*) на один розряд інтерпретується як операція ділення цілого додатного числа на 2. В результаті зсуву на один розряд вправо формується ціле значення частки від ділення на 2, а ознака перенесення містить значення залишку. При багаторозрядних зсувах ознака переносу формується за результатом останнього зсуву.

4) Цілі числа зі знаком

В процесорах 80x86 (Pentium) цілі числа зі знаком подаються в доповняльному коді. Крім того, вся адресна арифметика (формування ефективної адреси у випадку багатокомпонентної адреси, формування нового значення регістра IP при відносній адресації, формування фізичної адреси із логічної) в процесорах 80x86 (Pentium) виконується в доповняльному коді.

Позначимо через X_m доповняльний код числа X. Тоді

$$X_m = \begin{cases} X, \text{ якщо } X \geq 0 \\ 2^{k*8} - |X|, \text{ якщо } X < 0 \end{cases}$$

Діапазон значень доповняльного коду:

$$-2^{k*8-1} \leq X \leq +2^{k*8-1} - 1$$

Старший біт доповняльного коду визначає знак числа. Операція зміни знаку виконується за два кроки: на першому кроці виконується операція порозрядної логічної інверсії, а на другому – до результату додається 1.

Для ручної зміни знаку числа в доповняльному коді доречним є наступне правило: в коді числа справа (з молодших розрядів) пропустити всі 0 та першу 1, а в решті розрядів замінити 0 на 1, а 1 на 0.

У випадку відмінної розрядності чисел при виконанні операцій, розрядність числа з меншою розрядністю необхідно збільшити до розрядності числа з більшою розрядністю, не змінюючи ні знак, ні значення модуля числа. У випадку додатних чисел, додаткові старші розряди заповнюються значенням 0. У випадку від'ємних чисел, вони заповнюються значенням 1, тобто розширення полягає в заповненні старших розрядів значенням знакового розряду (команди *CBW - Convert Byte to Word*, *CWDE - Convert Word to DoubleWord*). Починаючи з процесорів Pentium (відповідно у всіх сучасних процесорах) реалізується універсальна команда знакового розширення *MOVSX.*

Операція додавання чисел зі знаком. Для додавання чисел із знаком в доповняльному коді можна використати операцію беззнакового додавання і, таким чином, одну й ту ж саму команду процесора (*ADD – ADDition*) як для додавання беззнакових чисел, так і для додавання чисел зі знаком. Але при виконанні операції додавання для чисел зі знаком необхідно враховувати додаткову ознаку - *ознаку переповнення* ***of*** (overflow flag) із регістра ознак, яка приймає значення 1, коли результат додавання виходить за межі діапазону та 0, коли результат додавання не виходить за межі діапазону (у випадку беззнакових чисел цю функцію виконує *ознака перенесення* ***cf***). Ознака переповнення записується в регістр ознак і далі може використовуватися в командах передачі управління за умовою. Крім того, ознака переповнення може бути джерелом внутрішнього переривання (виключення) для інформування, при необхідності, про помилку в обчисленнях.

При додаванні чисел зі знаком часто виникає потреба аналізувати знак результату. Для спрощення аналізу знаку в регістр ознак розміщують *ознаку знаку* ***sf*** *(sign flag)*, значення якої є копією значення старшого розряду результату. Ця ознака використовується в *командах передачі управління за умовою.* Команди передачі управління за умовою (див. далі) дозволяють проаналізувати ознаки, встановлені арифметичними (та деякими іншими) командами, та організувати правильний процес обробки у програмі.

Операція віднімання чисел зі знаком. Для віднімання чисел зі знаком у доповняльному коді можна використати операцію беззнакового віднімання і, таким чином, одну й ту ж саму команду процесора (*SUB – SUBtract*) як для віднімання беззнакових чисел, так і для віднімання чисел зі знаком. Операція віднімання може бути використана для порівняння чисел. Якщо для порівняння беззнакових чисел можна використовувати ознаку перенесення (наявність позики) та інколи додатково ознаку нуля, то для порівняння чисел зі знаком необхідно аналізувати співвідношення значень ознак переповнення та знаку (інколи додатково ознаку ***zf***). Сформовані (після виконання операції для чисел у доповняльному коді) ознаки використовуються в *командах передачі управління за умовою* (табл. 2-1.3).

Таблиця 2-1.3

Команди передачі управління за умовою для операндів зі знаком

Співвідношення між операндами	Значення ознак при відніманні (X - Y)	Найменування співвідношення	Мнемокод команди передачі управління за умовою
X > Y	(***of***=***sf***) і (***zf***=0)	Більше (або не менше і не дорівнює)	**JG** (або **JNLE**)
X ≥ Y	***of***=***sf***	Більше чи дорівнює (або не менше)	**JGE** (або **JNL**)
X = Y	***zf***=1	Дорівнює	**JE** (або **JZ**)
X ≠ Y	***zf***=0	Не дорівнює	**JNE** (або **JNZ**)
X ≤ Y	(***of***≠***sf***) чи (***zf***=1)	Менше чи дорівнює (або не більше)	**JLE** (або **JNG**)
X < Y	***of***≠***sf***	Менше (або не більше і не дорівнює)	**JL** (або **JNGE**, або **JC**)

Таким чином, тільки для співвідношень "дорівнює" та "не дорівнює" для чисел зі знаком в доповняльному коді і чисел без знаку команди передачі

управління за умовою співпадають. Для решти співвідношень операндів команди передачі за умовою відрізняються. Програміст і тільки програміст шляхом використання відповідних команд передачі управління за умовою визначає, які саме дані порівнювались. Це часто буває джерелом помилок при використанні у програмах команд передачі управління *JG\JNLE, JGE\JNL, JLE\JNG, JL\JNGE після порівняння чисел без знаку та команд JA\JNBE, JAE\JNB, JBE\JNA, JB\JNAE після порівняння чисел зі знаком.*

Операції множення і ділення. Операції відрізняються для чисел без знаку і чисел зі знаком у доповняльному коді, оскільки у другому випадку необхідно враховувати знаки операндів і формувати знак результату (команди множення для чисел без знаку *MUL - MULtiple* та для чисел зі знаком – *IMUL*; відповідно *DIV - DIVide* та *IDIV*).

Операції зсуву для чисел зі знаком. Для чисел зі знаком використовуються арифметичні зсуви.

Арифметичний зсув вліво (*SAL- Shift Arithmetic Left*). У випадку доповняльного коду арифметичний зсув вліво співпадає з лінійним зсувом вліво, тобто команда *SAL* еквівалентна команді *SHL.* І якщо програміст інтерпретує вміст бітового поля як число зі знаком, тоді після виконання операції зсуву на один розряд вліво потрібно аналізувати ознаку переповнення, а у випадку беззнакових чисел – ознаку перенесення. Необхідно мати на увазі, що у випадку багаторозрядних зсувів вліво ознака переповнення не визначена.

Арифметичний зсув вправо (*SAR - Shift Arithmetic Right*). При арифметичному зсуві вправо на один розряд у вивільнений старший розряд записується його попереднє (до зсуву) значення. Тим самим виконується операція ділення числа зі знаком на 2 з округленням в сторону -∞. При цьому ознака переповнення скидається в 0. Арифметичні зсуви на 2 і більше розрядів вправо визначаються як послідовність зсувів на один розряд. При багаторозрядному арифметичному зсуві вправо ознака переповнення вважається невизначеною.

Завдання операндів машинних команд

Можливі наступні варіанти завдання в машинних командах значень та місця розташування даних:

- операнди *за замовчуванням*;
- значення задаються *безпосередньо в команді*;
- операнди знаходяться *в регістрах*;
- операнди розташовуються *в пам'яті*.

Більшість команд процесорів 80x86 мають у своїй структурі адресну частину, яка у загальному випадку містить *байти режиму адресації modr/m*, *sib* та *зміщення в команді*.

Для операндів, які розташовані в пам'яті**,** адреса формується як сума двох складових: зсунутого на 4 біти вліво вмісту сегментного регістра та *зміщення у сегменті – ефективної адреси*, що у загальному випадку є сумою трьох компонент: *зміщення в команді* (задається безпосередньо), *бази* та *індексу*. База та індекс містяться в регістрах загального призначення **EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP**, які використовуються як адресні регістри**.** Індекс може мати множник 2,4 або 8, який визначає, на яке число необхідно помножити вміст 32-розрядного індексного регістра перед формуванням ефективної адреси (обмеження – регістр **ESP** не може задаватись із множником). Множник ефективно використовується для доступу до елементів масивів. Наприклад:

```
mov   eax, Ar[ebx + edi*4]
mov   edx, [ecx + edi]
mov   ebx, [eax + esi*2] + 8
```

Для 16-розрядних регістрів множник не задається, і як адресні регістри використовуються регістри **BX, SI, DI, BP.** Для формування ефективної адреси можуть використовуватись лише ці регістри, а також лише наступні їх пари: **BX+SI**, **BX+DI**, **BP+SI**, **BP+DI**. Наприклад:

```
mov   ax, [X + di]
mov   ax, [A + si - 4]
```

Докладніше про способи адресації див. у Лабораторній роботі №3.