**1. Регистры процессора. Типы регистров.**

**Регистром называется функциональный узел, осуществляющий приём, хранение и передачу информации. Регистры состоят из группы триггеров - это области высокоскоростной памяти, расположенные внутри процессора.**

**Типы регистров:**

* **Регистры данных (RAX/EAX/AX/AH/AL, RBX…, RCX…, RDX…)**
* **Индексные регистры (RDI/EDI/DI, RSI/ESI/SI)**
* **Регистры-указатели (RBP/EBP/BP, RSP/ESP/SP)**
* **Регистры состояния и управления (RFLAGS/EFLAGS/FLAGS, RIP/EIP/IP)**

**2. Регистры общего назначения.**

* **Аккумулятор (сумматор), EAX**
* **Базовый регистр, EBX**
* **Регистр счетчика, ECX**
* **Регистр данных, EDX**
* **Индекс источника, ESI**
* **Индекс приемника, EDI**
* **Указатель базы, EBP**
* **Указатель стека, ESP**

**3. Регистр флагов. Команды изменения флагов.**

**Регистр флагов RFLAGS/EFLAGS/FLAGS содержит информацию о текущем состоянии процессора и представляется в виде набора битовых флагов, изменяющихся и проверяемых независимо друг от друга В 16-разрядной версии он включает 6 флагов состояния и 3 флага управления процессором.**

**Типы флагов состояния:**

* **Флаг переноса, CF**
* **Вспомогательный флаг переноса, AF**
* **Флаг нуля, ZF**
* **Флаг знака, SF**
* **Флаг переполнения, OF**
* **Флаг четности, PF**

**Типы флагов управления процессором:**

* **Флаг трассировки, TF**
* **Флаг прерывания, IF**
* **Флаг направления, DF**

**Команды изменения флагов (где x - первая буква флага):**

* **Установить флаг, STx**
* **Сбросить флаг, CLx**

**4. Работа со стеком. Команды работы со стеком.**

**Для работы со стеком используются регистры SS, ESP и EBP. Содержимое SS является базой стека. В ESP хранится смещение вершины стека. Первоначально ESP инициализируется наибольшим смещением, которого может достигать стек, изменяется операциями включения и извлечения. Регистр EBP обычно используется для обращений к элементам стека.**

**Команды работы со стеком:**

* **Команда извлечения данных из стека, PUSH**
* **Команда помещения данных в стек, POP**

**Важно отметить, в стек заносятся только двойные слова. Кроме того, мы можем работать с элементами, находящимися в стеке, как с обычными данными в памяти. В частности, мы можем получить доступ к любому элементу стека, а не только к его вершине. Для этих целей активно используется регистр EBP.**

**5. Представление целых чисел в двоичной системе счисления.**

**Числа в могут занимать байт, слово или двойное слово (то есть 1, 2, или 4 байта соответственно), при этом старшие разряды числа хранятся в старших адресах.**

**Отрицательные числа хранятся в дополнительном коде (двоичное дополнение):**

* **Запись двоичного представления модуля числа**
* **Инверсия каждого бита**
* **Добавление к результату единицы**

**Различают неупакованный и упакованный форматы хранения двоично-десятичных чисел. При хранении в неупакованном формате каждая цифра занимает один байт, в упакованном – в одном байте хранятся две цифры. Длина числа и положение десятичной точки определяются программистом.**

**Числа с плавающей точкой обрабатываются сопроцессором. Данные в этом случае задаются в виде знака, мантиссы и характеристики, при этом:**

* **Бит знака устанавливается в 1, если число отрицательное**
* **Старший бит мантиссы равен 1, если число не равно 0**

**6. Адресация данных в машинных командах. Типы адресации.**

**Адресация – информация о том, где находятся обрабатываемые командой данные. Возможно следующее расположение данных:**

* **в самой команде (непосредственная адресация)**
* **в регистрах (регистровая адресация)**
* **в памяти (адресация в памяти, может быть прямой и косвенной)**

**Эффективный адрес получается как сумма адресов, хранящихся**

* **в самой команде**
* **в регистрах RBX/EBX/BX или RBP/EBP/BP**
* **в регистрах RSI/ESI/SI или RDI/EDI/DI**

**7. Команды сложения и вычитания.**

**Команды сложения и вычитания (команды вычитания реализованы через команды сложения):**

* **ADD, складывает оба операнда, результат помещается в первый операнд**
* **ADC, складывает оба операнда и флаг CF, результат помещается в первый операнд**
* **SUB, вычитает из первого операнда второй, результат помещается в первый операнд**
* **SBB, вычитает из первого операнда второй и флаг CF, результат - в первый операнд**
* **INC, увеличивает операнд на единицу, флаг CF не изменяется**
* **DEC, уменьшает операнд на единицу, флаг CF не изменяется**
* **CMP, вычисляет разность операндов, результат не сохраняется, выставляются флаги**
* **NEG, изменяет знак операнда (Если операнд отрицательный и максимальный по модулю, то выполнить эту команду нельзя - не существует равного положительного числа, в таком случае выставляется флаг OF)**

**8. Операторы преобразования длины.**

**Операторы преобразования длины позволяют явно указать или переопределить длину данных.**

**Виды операторов:**

* **BYTE PTR - 1 байт**
* **WORD PTR - 2 байта**
* **DWORD PTR - 4 байта**
* **QWORD PTR - 8 байт**

**Тип операнда может быть любым, тип результата – такой же, как и тип операнда. Важно отметить, что не выполняется при регистровой адресации.**

**Существуют и команды преобразования чисел со знаком:**

* **СBW, преобразует байт из регистра AL в слово регистра AX**
* **CWD, преобразует слово регистра AX в двойное слово DX:AX**
* **CWDE, преобразует слово регистра AX в двойное слово EAX**
* **CDQ, преобразует двойное слово EAX в учетверенное слово EDX:EAX**

**9. Команды пересылки данных.**

**Команда MOV – команда пересылки данных: переносит содержимое второго операнда в первый операнд, содержимое второго операнда не меняется.**

**Ограничения, накладываемые командой MOV на комбинации операндов:**

* **длины операндов должны быть равны**
* **первый операнд не может быть непосредственным**
* **запрещена пересылка память – память**
* **запрещена пересылка в регистры CS и EIP (этим должны заниматься команды передачи управления)**
* **пересылка в сегментные регистры возможна только из регистров общего назначения или из памяти.**

**Команда XCHG меняет местами данные первого и второго операндов.**

**Команда LEA загружает в регистр вычисленное во время выполнения адресное выражение. Если выражение может быть вычислено во время компиляции, заменяется машинной командой MOV. Например, команде LEA BX, X[BX] нет эквивалентной записи команды MOV.**

**10. Классификация команд управления программным потоком.**

**Категории команд:**

* **Переходы (условные, безусловные)**
* **Вызовы подпрограмм**
* **Прерывания**

**Особенности работы команд управления программным потоком:**

* **Безусловные переходы (JMP) изменяют содержимое регистров EIP и, возможно, CS (в зависимости от информации, хранящейся в команде)**
* **Условные переходы (Jxxx) выполняются при выполнении определенного условия**
* **Вызовы подпрограмм (CALL) сохраняют старые значения EIP и CS в стеке**
* **Выходы из подпрограмм (RET) извлекают старые значения EIP и CS из стека**
* **Прерывания (INT) сохраняют старые значения EIP и CS в стеке**
* **Выходы из прерываний (IRET) извлекают старые значения EIP и CS из стека**

**11. Условные переходы.**

**Условные переходы Jxxx принимают решение о переходе в зависимости от выполнения определенных условий:**

* **Пустоты регистра CX/ECX**
* **Состояния флагов ZF, OF, CF, SF**
* **Результатов сравнения целых знаковых и беззнаковых целых чисел**

**12. Команды условной пересылки.**

**Команды условной пересылки CMOVxxx появились в процессорах PentiumPro и Pentium II. Они копируют содержимое источника в приемник, если удовлетворяется то или иное условие. Коды условий аналогичны кодам в командах условного перехода (например, CMOVLE или CMOVC).**

**13. Организация циклов с помощью переходов.**

| **Организация цикла WHILE метка:   CMP оп1, оп2   Jxx метка\_выхода**  **…**  **JMP метка**  **метка\_выхода:** | **Организация цикла DO WHILE**  **метка:**  **…**  **CMP оп1, оп2**  **Jxx метка** |
| --- | --- |

**14. Организация циклов с помощью команды LOOP и ее модификаций.**

| **Организация цикла LOOP**  **MOV ECX, N**  **метка:**  **…**  **LOOP метка** | **Организация защищенного цикла LOOP**  **MOV ECX, N**  **JECXZ метка\_выхода**  **метка:**  **…**  **LOOP метка**  **метка\_выхода:** |
| --- | --- |

**15. Команды умножения целых чисел.**

**Команды умножения:**

* **MUL, умножение беззнаковых чисел**
* **IMUL, умножение знаковых чисел**

| **Длина операнда** | **Второй операнд** | **Результат** |
| --- | --- | --- |
| **Байт** | **AL** | **AX** |
| **Слово** | **AX** | **DX, AX** |
| **Двойное слово** | **EAX** | **EDX, EAX** |

**Если размер результата превышает размер множителей, устанавливаются флаги OF, CF.**

**16. Команды деления целых чисел.**

**Команды деления:**

* **DIV, деление беззнаковых чисел**
* **IDIV, деление знаковых чисел**

| **Длина операнда** | **Делимое** | **Частное** | **Остаток** |
| --- | --- | --- | --- |
| **Байт** | **AX** | **AL** | **AH** |
| **Слово** | **DX, AX** | **AX** | **DX** |
| **Двойное слово** | **EDX, EAX** | **EAX** | **EDX** |

**17. Команды для работы с битами.   
Команды AND, OR, XOR выполняют соответствующие логические операции над битами первого и второго операндов, результат помещается в первый операнд. Команда NOT инвертирует биты операнда и помещает результат в него же. Команда TEST вычисляет значение конъюнкции операндов, результат не сохраняет и выставляет флаги (подобно CMP). Все команды также выставляют флаг ZF, если результат содержит только нули.**

**18. Команды линейного сдвига.**

**Алгоритм линейного сдвига:**

* **Очередной «выдвигаемый» бит устанавливает флаг CF (сдвигается во флаг CF).**
* **Бит, появляющийся с другого конца операнда, имеет значение 0.**
* **При сдвиге очередного бита он переходит во флаг CF, при этом значение предыдущего сдвинутого бита теряется.**

**Команды линейного сдвига делятся на два подтипа: линейного и арифметического сдвига.**

**Арифметический сдвиг вправо сохраняет бит знака.**

**19. Команды циклического сдвига.**

**Команды циклического сдвига делятся на два подтипа: простого сдвига и сдвига через флаг CF.**

**Значения сдвигаемых блоков сохраняются в флаг CF и, в случае с простым сдвигом, переносятся в противоположный конец битового блока, а в случае с сдвигом через флаг - место освобожденного бита занимает бит флага CF.**

**20. Доступ к элементам одномерных массивов.**

**Массив располагается в непрерывном блоке памяти размером SizeArray \* sizeof(TypeArray) байт.**

**Нумерация элементов в блоке идет слева направо, например для 3-его элемента int-массива Array необходимо обращение Array[2] или Array+8. Идентификатор массива может использоваться как указатель на элемент массива с индексом 0.**

**21. Доступ к элементам двумерных массивов.**

**Аналогичное расположение в памяти непрерывным блоком. Для массива Array[R][C] обращение к элементу A[i][j] происходит через Array+i\*(C\*K)+j\*K = Array+(i\*C+j)\*K. Грубо говоря, вместо привычного элемента i одномерного массива, в ячейке i находится еще один линейно расположенный непрерывный подмассив ряда i длиной С.**

**22. Строковые команды. Префиксы повторения в строковых командах.**

**Существуют следующие строковые команды (где [X] - B, W, D в зависимости от элемента строки):**

* **MOVS[X], пересылка текущего элемента строки-источника(DS) в строку-приемник(ES)**
* **CMPS[X], сравнение текущего элемента строки-источника и строки-приемника, устанавливая флаги (подобно CMP)**
* **LODS[X], пересылка содержимого строки-источника в аккумулятор (EAX/AX/AL)**
* **STOS[X], пересылка содержимого аккумулятора в строку-приемник**
* **SCAS[X], сравнение содержимого аккумулятора с текущим элемент строки-приемника**
* **CLD, сброс флага DF (Direction Flag)**
* **STD, установка флага DF (Direction Flag)**

**Префиксы повторения - REP, REPE/REPZ, REPNE/REPNZ - работают с регистром CX/ECX в качестве счетчика повторений (подобно LOOP, только проверка ECX на 0 происходит прежде чем выполнение команды, у LOOP - наоборот).**

**23. Механизм вызова процедур. Команды для работы с процедурами.**

**Для оформления процедур используется конструкция:**

**Имя\_процедуры PROC [NEAR|FAR]**

**…**

**RET**

**Имя\_процедуры ENDP**

**Важно понимать, что директивы PROC/ENDP не генерируют никакого кода.**

**Команды для работы с процедурами:**

* **CALL адрес, где адрес - точка входа в процедуру, можно задать спецификатор NEAR/FAR**
* **RET, выход из процедуры**

**Различают ближний (NEAR) и дальний (FAR) вызовы - решено аппаратно, а также прямой и косвенный вызовы - решено программно.  
  
24. Способы передачи параметров в процедуры.**

**В архитектуре процессоров x86 нет никаких аппаратных решений по механизму передачи параметров.**

**Параметры могут быть переданы:**

* **через общие области памяти**
* **через регистры**
* **через стек**

**25. Проблемы передачи параметров через стек и способы их решения.**

**Суть метода – вызывающая программа перед выполнением команды CALL заносит в стек либо значения фактических параметров, либо их адреса. Процедура извлекает из стека полученную информацию без использования команд POP, используя тот факт, что к содержимому стека можно обращаться, как к любому другому участку памяти. Достоинство – количество параметров и их размер практически неограниченны. Недостаток – необходимо продумывать механизмы помещения в стек и извлечения из стека.**

**При передаче данных через стек необходимо решить следующие вопросы**

* **В каком порядке передавать параметры**
* **Какая программа должна очищать стек**

**Возможны два варианта очистки:**

* **Процедура очищает стек в момент выхода (выполнения команды RET)**
* **Вызывающая программа очищает стек после того, как она вновь получила управление**

**26. Использование регистра EBP для доступа к параметрам.**

**Стек передачи данных:**

* **[ESP], Старый EBP**
* **[EBP+4], Адрес возврата**
* **[EBP+8], Первая переменная**
* **[EBP+12], Вторая переменная**
* **и т.д.**

**Необходима организация кода:**

**PUSH EBP**

**MOV EBP ESP**

**…**

**MOV ESP, EBP**

**POP EBP**

**27. Организация работы с локальными переменными.**

**Пример организации работы с локальными данными:**

**FOO PROC**

**JMP @@M\_**

**@@CRLF DB 13, 10, ‘$’**

**@@M\_:**

**…**

**RET**

**FOO ENDP**

**28. Вызов ассемблерных процедур из программ на C++.**

**В файле \*.asm до директивы PROC необходимо:**

* **Включить директиву выбора CPU для распознавания отдельных команд, например: .386 или .486 для адресации по 32-битным регистрам (всех команд вплоть до указанного процессора)**
* **Объявить имя процедуры с директивой PUBLIC**
* **Включить модель памяти flat (.model flat)**

**При написании текста процедуры соблюдать требования:**

* **параметры являются 32-разрядными**
* **при доступе к параметрам учитывать их порядок**
* **при возврате управления при необходимости очищать стек от параметров**

**В файле \*.cpp необходимо:**

* **Описать прототип ассемблерной функциив виде extern "С" тип\_возврата тип\_соглашения имя(параметры)**

**Важно помнить что компиляторы С/С++ искажают декорацию (имена) функций.**

**29. Соглашения о связях.**

**По умолчанию тип соглашения: \_\_CDECL.**

|  | **CDECL** | **STDCALL** | **FASTCALL** |
| --- | --- | --- | --- |
| **Передача параметров** | **Справа налево** | **Справа налево** | **Слева направо** |
| **Очистка стека** | **Вызывающая программа** | **Процедура (RET)** | **Вызывающая программа** |
| **Регистры для пар-ов** | **Нет** | **Нет** | **ECX, EDX** |
| **Декорация** | **\_NAME** | **\_NAME@NUM** | **@NAME@NUM** |

**30. ФИО Преподавателя и название учебной дисциплины.**

**Конах Валентина Владимировна, Машинно-Ориентированное Программирование.**