Ассемблер: история, цели и возможности

Начиная изучать программирование, когда пытаешься следить за трендами в среде профессиональных программистов, очень сильно настораживает распространение очень далеких от аппаратного обеспечения языков программирования. Вернее сама распространенность этих языков (Java, C#, PHP) вполне объяснима: если нет критической необходимости с высокой производительности, то зачем усложнять себе жизнь? Ведь можно просто использовать язык программирования, который содержит необходимые защитные механизмы, которые предотвращают появления большинства трудноуловимых проблем, тогда задача программиста сводится только к выражению концепций на достаточно высоком уровне абстракции.

Настораживает, то, что указанный образ мышления заставляет думать, что аппаратный уровень знать не обязательно. Во всяком случае, мне кажется что есть такая тенденция, именно поэтому, я и пишу о языке ассемблера, хотя сам только начал ему учиться.

Язык ассемблера или автокод, был разработан в 1949 г. как способ записи машинных команд, которые непосредственно транслируются в машинный код.

mov DST,SRC - загpузка в DST значения из SRC

push SRC - запись SRC в стек

pop DST - загpузка слова из стека в DST

inc DST - увеличение DST на единицу

add DST,SRC - сложение DST и SRC

div SRC - деление на значение в SRC

and DST,SRC - логическое умножение DST и SRC

jz LBL - условный пеpеход, если ноль

jmp LBL - безусловный пеpеход (LBL - метка)

call LBL - вызов подпpогpаммы

int NUM - вызов подпpогpаммы обpаботки пpеpывания

ret - возвpат из подпpогpаммы

iret - возвpат из подпpогpаммы обpаботки пpеpывания

В первом столбце указаны команды.

Язык ассемблера зависим от конкретной архитектуры процессора, и столько же этих языков сколько существует процессоров и встраиваемых систем.

Тем не менее и языка высокого уровня оказали влияние и на язык ассеблера, в частности у него появились такие опции как:

1. Макрокоманды;
2. Выражения;
3. Средства обеспечения модульности программ.

Язык ассемблера используется обычно в случаях когда необходима:

* Оптимизация критичных к скорости участков программ в программах на языках высокого уровня, таких как [C++](https://ru.wikipedia.org/wiki/C%2B%2B) или [Pascal](https://ru.wikipedia.org/wiki/Pascal" \o "Pascal). Это особенно актуально для [игровых приставок](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0), имеющих фиксированную производительность, и для [мультимедийных](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0" \o "Мультимедиа)[кодеков](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D0%B5%D0%BA), которые стремятся делать менее ресурсоёмкими и более быстрыми.
* Создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) (ОС) или их компонентов. В настоящее время подавляющее большинство ОС пишут на более высокоуровневых языках (в основном на [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) — языке высокого уровня, который специально был создан для написания одной из первых версий [UNIX](https://ru.wikipedia.org/wiki/UNIX)). Аппаратно зависимые участки кода, такие как [загрузчик ОС](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D1%87%D0%B8%D0%BA_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B), уровень [абстрагирования от аппаратного обеспечения](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B9_%D0%B0%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D1%85_%D0%B0%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D0%B9) (hardware abstraction layer) и ядро, часто пишутся на языке ассемблера. Фактически, ассемблерного кода в ядрах [Windows](https://ru.wikipedia.org/wiki/Windows" \o "Windows) или [Linux](https://ru.wikipedia.org/wiki/Linux_(%D1%8F%D0%B4%D1%80%D0%BE)" \o "Linux (ядро)) совсем немного, поскольку авторы стремятся обеспечить [переносимость](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) и [надёжность](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C), но, тем не менее, он там присутствует. Некоторые любительские ОС, такие как [MenuetOS](https://ru.wikipedia.org/wiki/MenuetOS" \o "MenuetOS) и [KolibriOS](https://ru.wikipedia.org/wiki/KolibriOS" \o "KolibriOS), целиком написаны на языке ассемблера. При этом MenuetOS и KolibriOS помещаются на дискету и содержат графический многооконный интерфейс.
* Программирование [микроконтроллеров](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80) (МК) и других встраиваемых процессоров. По мнению профессора [Таненбаума](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BD%D0%B5%D0%BD%D0%B1%D0%B0%D1%83%D0%BC,_%D0%AD%D0%BD%D0%B4%D1%80%D1%8E" \o "Таненбаум, Эндрю), развитие МК повторяет историческое развитие компьютеров новейшего времени[[9]](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0#cite_note-9). Сейчас (2013 г.) для программирования МК весьма часто применяют язык ассемблера (хотя и в этой области широкое распространение получают языки вроде Си). В МК приходится перемещать отдельные [байты](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82) и [биты](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) между различными [ячейками памяти](https://ru.wikipedia.org/wiki/%D0%AF%D1%87%D0%B5%D0%B9%D0%BA%D0%B0_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%B8). Программирование МК весьма важно, так как, по мнению Таненбаума, в автомобиле и квартире современного цивилизованного человека в среднем содержится 50 микроконтроллеров[[10]](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0#cite_note-10).
* Создание [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80). Драйверы (или их некоторые программные модули) программируют на языке ассемблера. Хотя в настоящее время драйверы также стремятся писать на языках высокого уровня (на высокоуровневом языке много проще написать надёжный драйвер) в связи с повышенными требованиями к надёжности и достаточной производительностью современных процессоров (быстродействие обеспечивает временно́е согласование процессов в устройстве и процессоре) и достаточным совершенством компиляторов с языков высокого уровня (отсутствие ненужных пересылок данных в сгенерированном коде), подавляющая часть современных драйверов пишется на языке ассемблера. Надёжность для драйверов играет особую роль, поскольку в [Windows NT](https://ru.wikipedia.org/wiki/Windows_NT" \o "Windows NT) и [UNIX](https://ru.wikipedia.org/wiki/UNIX) (в том числе в [Linux](https://ru.wikipedia.org/wiki/Linux" \o "Linux)) драйверы работают в режиме ядра системы. Одна тонкая ошибка в драйвере может привести к краху всей системы.
* Создание [антивирусов](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D1%82%D0%B8%D0%B2%D0%B8%D1%80%D1%83%D1%81%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) и других защитных программ.
* Написание [трансляторов](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) языков программирования.
* .386
* .model flat, stdcall
* option casemap:none
* include \masm32\include\windows.inc
* include \masm32\include\kernel32.inc
* includelib \masm32\lib\kernel32.lib
* .data
* msg db "Hello, world", 13, 10
* len equ $-msg
* .data?
* written dd ?
* .code
* start:
* push -11
* call GetStdHandle
* push 0
* push OFFSET written
* push len
* push OFFSET msg
* push eax
* call WriteFile
* push 0
* call ExitProcess
* end start

Любой код на языке ассемблера состоит из сегментов:

1. Кода;
2. Данных;
3. Стека;

А также обязательных элементов:

1. Указание архитектуры: .386
2. Используемая модель памяти: flat
3. Соглашения о вызовах: stdcall

Присутствуют возможности подключения библиотек, а также заголовочных файлов: .lib, .inc

Исполняемый код организован по принципу меток.

Ведущая метка, это метка, которая указана после директивы END.

Возможности языка ассемблера безграничны. Он один охватывает все возможные способы выполнения программ на всех возможных языках, поскольку любой исходный код можно повторить на ассемблере.

#include <iostream>

int main()

{

std::cout << “Hello world!!!” << ‘\n’;

char c;

std::cin >> c;

}

7714db5a 5b pop ebx

7714db5b e8402dfcff call ntdll!\_\_security\_check\_cookie (771108a0)

7714db60 8be5 mov esp,ebp

7714db62 5d pop ebp

7714db63 c3 ret

7714db64 6a0c push 0Ch

7714db66 6840001a77 push offset ntdll!ResCSegmentValidateHeader+0x157a (771a0040)

7714db6b e83cb5fdff call ntdll!\_SEH\_prolog4 (771290ac)

7714db70 6a00 push 0

7714db72 6a01 push 1

7714db74 8d45e7 lea eax,[ebp-19h]

7714db77 50 push eax

7714db78 6a11 push 11h

7714db7a 6afe push 0FFFFFFFEh

7714db7c e84f0efcff call ntdll!NtQueryInformationThread (7710e9d0)

7714db81 85c0 test eax,eax

7714db83 781b js ntdll!LdrpDoDebuggerBreak+0x3c (7714dba0)

7714db85 807de700 cmp byte ptr [ebp-19h],0

7714db89 7515 jne ntdll!LdrpDoDebuggerBreak+0x3c (7714dba0)

7714db8b 8365fc00 and dword ptr [ebp-4],0

7714db8f cc int 3

7714db90 eb07 jmp ntdll!LdrpDoDebuggerBreak+0x35 (7714db99)

7714db92 33c0 xor eax,eax

7714db94 40 inc eax

7714db95 c3 ret

7714db96 8b65e8 mov esp,dword ptr [ebp-18h]

7714db99 c745fcfeffffff mov dword ptr [ebp-4],0FFFFFFFEh

7714dba0 e84cb5fdff call ntdll!\_SEH\_epilog4 (771290f1)

7714dba5 c3 ret

7714dba6 64a130000000 mov eax,dword ptr fs:[00000030h]

7714dbac 33c9 xor ecx,ecx

7714dbae 890d54861b77 mov dword ptr [ntdll!LdrpForkActiveLock (771b8654)],ecx

7714dbb4 890d58861b77 mov dword ptr [ntdll!LdrpForkConditionVariable (771b8658)],ecx

7714dbba 8808 mov byte ptr [eax],cl

7714dbbc 384802 cmp byte ptr [eax+2],cl

7714dbbf 7405 je ntdll!LdrpForkProcess+0x20 (7714dbc6)

7714dbc1 e89effffff call ntdll!LdrpDoDebuggerBreak (7714db64)

7714dbc6 33c0 xor eax,eax

7714dbc8 c3 ret

7714dbc9 8bff mov edi,edi

7714dbcb 55 push ebp

7714dbcc 8bec mov ebp,esp

7714dbce 83e4f8 and esp,0FFFFFFF8h

7714dbd1 81ec70010000 sub esp,170h

7714dbd7 a150b31b77 mov eax,dword ptr [ntdll!\_\_security\_cookie (771bb350)]

7714dbdc 33c4 xor eax,esp

7714dbde 8984246c010000 mov dword ptr [esp+16Ch],eax

7714dbe5 56 push esi

7714dbe6 8b3500921b77 mov esi,dword ptr [ntdll!g\_pfnApphelpCheckModuleProc (771b9200)]

7714dbec 57 push edi

7714dbed 6a16 push 16h

7714dbef 58 pop eax

7714dbf0 6689442410 mov word ptr [esp+10h],ax

7714dbf5 8bf9 mov edi,ecx

7714dbf7 6a18 push 18h

7714dbf9 58 pop eax

7714dbfa 6689442412 mov word ptr [esp+12h],ax

7714dbff 8d442470 lea eax,[esp+70h]

7714dc03 8944246c mov dword ptr [esp+6Ch],eax

7714dc07 33c0 xor eax,eax

7714dc09 c744241480460a77 mov dword ptr [esp+14h],offset ntdll!`string' (770a4680)

7714dc11 c744246800000001 mov dword ptr [esp+68h],1000000h

7714dc19 6689442470 mov word ptr [esp+70h],ax

7714dc1e 85f6 test esi,esi

7714dc20 7429 je ntdll!LdrpGetProcApphelpCheckModule+0x82 (7714dc4b)

7714dc22 8b153003fe7f mov edx,dword ptr [SharedUserData+0x330 (7ffe0330)]

7714dc28 8bc2 mov eax,edx

7714dc2a 6a20 push 20h

7714dc2c 83e01f and eax,1Fh

7714dc2f 59 pop ecx

7714dc30 2bc8 sub ecx,eax

7714dc32 d3ce ror esi,cl

7714dc34 33f2 xor esi,edx

7714dc36 8937 mov dword ptr [edi],esi

7714dc38 7407 je ntdll!LdrpGetProcApphelpCheckModule+0x78 (7714dc41)

7714dc3a 33f6 xor esi,esi

7714dc3c e91c010000 jmp ntdll!LdrpGetProcApphelpCheckModule+0x194 (7714dd5d)

7714dc41 be010000c0 mov esi,0C0000001h

7714dc46 e912010000 jmp ntdll!LdrpGetProcApphelpCheckModule+0x194 (7714dd5d)

7714dc4b 8d542410 lea edx,[esp+10h]

7714dc4f 8d4c2468 lea ecx,[esp+68h]

7714dc53 e85d5df7ff call ntdll!LdrpBuildSystem32FileName (770c39b5)

7714dc58 8bf0 mov esi,eax

7714dc5a 85f6 test esi,esi

7714dc5c 0f88e8000000 js ntdll!LdrpGetProcApphelpCheckModule+0x181 (7714dd4a)

7714dc62 8d442418 lea eax,[esp+18h]

7714dc66 ba01400000 mov edx,4001h

7714dc6b 50 push eax

7714dc6c 33c9 xor ecx,ecx

7714dc6e e8c9adf8ff call ntdll!LdrpInitializeDllPath (770d8a3c)

7714dc73 8d44240c lea eax,[esp+0Ch]

7714dc77 50 push eax

7714dc78 6a01 push 1

7714dc7a 6a00 push 0

7714dc7c 8d542424 lea edx,[esp+24h]

7714dc80 8d4c2474 lea ecx,[esp+74h]

7714dc84 e813d9f9ff call ntdll!LdrpLoadDll (770eb59c)

7714dc89 807c246400 cmp byte ptr [esp+64h],0

7714dc8e 8bf0 mov esi,eax

7714dc90 7409 je ntdll!LdrpGetProcApphelpCheckModule+0xd2 (7714dc9b)

7714dc92 ff742418 push dword ptr [esp+18h]

7714dc96 e81590f9ff call ntdll!RtlReleasePath (770e6cb0)

7714dc9b 85f6 test esi,esi

7714dc9d 7939 jns ntdll!LdrpGetProcApphelpCheckModule+0x10f (7714dcd8)

7714dc9f a190571b77 mov eax,dword ptr [ntdll!LdrpDebugFlags (771b5790)]

7714dca4 a803 test al,3

7714dca6 7429 je ntdll!LdrpGetProcApphelpCheckModule+0x108 (7714dcd1)

7714dca8 56 push esi

7714dca9 8d44246c lea eax,[esp+6Ch]

7714dcad 50 push eax

7714dcae 6824f60a77 push offset ntdll!`string' (770af624)

7714dcb3 6a00 push 0

7714dcb5 6864f60a77 push offset ntdll!`string' (770af664)

7714dcba 68eb0a0000 push 0AEBh

7714dcbf 6850680a77 push offset ntdll!`string' (770a6850)

7714dcc4 e8f4d1ffff call ntdll!LdrpLogDbgPrint (7714aebd)

7714dcc9 83c41c add esp,1Ch

7714dccc a190571b77 mov eax,dword ptr [ntdll!LdrpDebugFlags (771b5790)]

7714dcd1 a810 test al,10h

7714dcd3 7475 je ntdll!LdrpGetProcApphelpCheckModule+0x181 (7714dd4a)

7714dcd5 cc int 3

7714dcd6 eb72 jmp ntdll!LdrpGetProcApphelpCheckModule+0x181 (7714dd4a)

7714dcd8 8b44240c mov eax,dword ptr [esp+0Ch]

7714dcdc 81483400010000 or dword ptr [eax+34h],100h

7714dce3 8b44240c mov eax,dword ptr [esp+0Ch]

7714dce7 8b4018 mov eax,dword ptr [eax+18h]

7714dcea a3b87b1b77 mov dword ptr [ntdll!g\_pShimEngineModule (771b7bb8)],eax

7714dcef e8e159f7ff call ntdll!LdrpGetShimEngineInterface (770c36d5)

7714dcf4 8bf0 mov esi,eax

7714dcf6 85f6 test esi,esi

7714dcf8 792a jns ntdll!LdrpGetProcApphelpCheckModule+0x15b (7714dd24)

7714dcfa a190571b77 mov eax,dword ptr [ntdll!LdrpDebugFlags (771b5790)]

7714dcff a803 test al,3

7714dd01 74ce je ntdll!LdrpGetProcApphelpCheckModule+0x108 (7714dcd1)

7714dd03 56 push esi

7714dd04 684cf50a77 push offset ntdll!`string' (770af54c)

7714dd09 6a00 push 0

7714dd0b 6864f60a77 push offset ntdll!`string' (770af664)

7714dd10 68f60a0000 push 0AF6h

7714dd15 6850680a77 push offset ntdll!`string' (770a6850)

7714dd1a e89ed1ffff call ntdll!LdrpLogDbgPrint (7714aebd)

7714dd1f 83c418 add esp,18h

7714dd22 eba8 jmp ntdll!LdrpGetProcApphelpCheckModule+0x103 (7714dccc)

7714dd24 8b153003fe7f mov edx,dword ptr [SharedUserData+0x330 (7ffe0330)]

7714dd2a 8bc2 mov eax,edx

7714dd2c 83e01f and eax,1Fh

7714dd2f 6a20 push 20h

7714dd31 59 pop ecx

7714dd32 2bc8 sub ecx,eax

7714dd34 a100921b77 mov eax,dword ptr [ntdll!g\_pfnApphelpCheckModuleProc (771b9200)]

7714dd39 d3c8 ror eax,cl

7714dd3b 33c2 xor eax,edx

7714dd3d 8907 mov dword ptr [edi],eax

7714dd3f 7404 je ntdll!LdrpGetProcApphelpCheckModule+0x17c (7714dd45)

7714dd41 33f6 xor esi,esi

7714dd43 eb05 jmp ntdll!LdrpGetProcApphelpCheckModule+0x181 (7714dd4a)

7714dd45 be010000c0 mov esi,0C0000001h

7714dd4a 8d442470 lea eax,[esp+70h]

7714dd4e 3b44246c cmp eax,dword ptr [esp+6Ch]

7714dd52 7409 je ntdll!LdrpGetProcApphelpCheckModule+0x194 (7714dd5d)

7714dd54 ff74246c push dword ptr [esp+6Ch]