Půlsemestrální písemka IMP dne 10. 11. 2017

Pár poznámek k Vašim odpovědím

Příklad 1 – hodnoty v zásobníku, SP

Cílem bylo vyzkoušet zejména pochopení
i) způsobu nastavení ukazatele zásobníku (SP) po resetu ARM, mechanismu růstu zásobníku a vývoje SP při vkládání dat na zásobník,
ii) vlivu „endianness“ na ukládání vícebytových dat do paměti.

Je třeba si uvědomit, jak funguje zásobník, ukazatel zásobníku SP, kde se bere jeho obsah. Vědět, co je vlastně obsahem jedné buňky paměti (jedné adresy) – že je to jeden bajt (osmibitové slovo) a jak je v paměti uloženo 32 bitové slovo – že je rozloženo na 4 bajty a kam který patří (endianness). Toto jsou znalosti převážně obecného charakteru, s drobnými implementačními rozdíly platné pro jakýkoliv číslicový počítač.

Největší potíže činilo nastavit správnou hodnotu SP po resetu, demonstrovat správný vývoj hodnoty SP při vkládání dat na zásobník a uložit data na zásobník dle zadaného typu endianness.

Příklad 2 – funkce přijetí/vyslání znaku prostřednictvím UART

Cílem bylo zjistit, zda rozumíte mechanismu komunikace s periferním zařízením pomocí tzv. pollingu, tj. softwarového sledování příznaku a umíte jej zapsat. Šlo doslova o to, co se procvičilo v první laboratorní úloze.

Často se vyskytovaly problémy se zápisem jednoduchých konstrukcí v C. Buď někteří neumí C nebo nerozumí tomu, co se přesně má udělat.

Například: if (test_priznaku) UART0_D = ch;

Z takového řešení bolí hlava – tisíckrát Vám to data pošle, ale někdy ne. Proč asi? Nebo naopak, neposílá to (při opačné podmínce), takže to data zahazuje a program to netuší. Při čekání na příznak je třeba cyklus! Jen jednorázový test nestačí, protože nikdo nezaručí, že příznak bude nastaven právě v okamžiku, kdy program přijde sem. To je princip konstrukce, která byla zkoušena – jde o synchronizaci běhu programu s vnější událostí.

Někteří napsali sice cyklus, ale nedokázali napsat správně tělo cyklu. Například:

while (test_priznaku) UART_D = ch;

případně ještě hůř while (test_priznaku) { UART_D = ch;} nebo {return UART_D;} dle skupiny.

Toto znamená co? Že se bude posílat znak tak dlouho, dokud nebude splněn test? Ale má se poslat jen jednou. Pokud bude test splněn rovnou při zavolání funkce, nepošle se nic. To je přece špatně. Při čekání na příznak má být tělo cyklu prázdné – program nedělá nic, než že čeká na příznak. Až je tento nastaven pak teprve program pokračuje manipulací s daty. Tato manipulace je již za cyklem, tedy vně těla cyklu.

Různé konstrukce s “break” a “continue” v cyklech nejsou moc čistým řešením. Možná by to fungovalo, ale chtělo se mi v takových případech říct: doufám, že se budete živit hardwarem.

Dále – jakou hodnotu má mít maska, když má být v 8 bitovém čísle nejvyšší bit nastaven na 1 a ostatní 0? 0x80 nebo 0x01? Tento předmět předpokládá, že takové základy znáte již od prvního ročníku. Kdo v tom tápe, zkuste si to prosím ujasnit nejpozději do zkoušky.

Příklad 3 – nastavení GPIO pinů pro jednoduché vstupy a výstupy.

Příklad měl ověřit, zda vidíte souvislost mezi zapojením MCU v aplikaci (vestavěním počítače do aplikace - vestavný systém) a jeho SW konfigurací a máte představu, jak taková konfigurace vypadá (že jde o nastavování bitů v registrech a co asi lze nastavit). Základní a nejpoužívanější jsou "běžné" číslicové vstupy/výstupy (General Purpose Input/Output - GPIO). Měli byste vědět, "o co jde", jaká obvykle jsou a k čemu slouží základní volby spojené s pinem MCU (směr pinu, pull-up resistor atd.), zbytek lze vyčíst z manuálu – významy konkrétních bitů. Ty byly k dispozici v zadání.

U mnohých se jako kámen úrazu nečekaně ukazuje šestnáctková soustava. Šestnáctková soustava se používá jen proto, že převod z dvojkové do šestnáctkové je velmi snadný (přitom zápis v šestnáctkové je přehlednější a čitelnější než ve dvojkové, je mnohem menší pravděpodobnost chyby). Převod lze provádět bez námahy "z hlavy" – bez použití kalkulačky atd. Pokud byste chtěli být pouze laickými uživateli počítačů, lze tuto neznalost omluvit. Uživatelé ale nemusí mít VŠ diplom z IT, že.

Příklad 4 – tabulka přerušovacích vektorů

Cílem bylo prověřit zejména pochopení základních
i) vazeb mezi tabulkou vektorů přerušení, adresou/významem vektoru a číslem IRQ,
ii) mechanismů souvisejících s přerušovacím podsystémem (nulování příznaků a od/maskování přerušení, zpracování obsluh neobsloužených přerušení v pořadí daném prioritou).

Přerušovací systém slouží k tomu, aby se v případě nějaké události (o níž nelze dopředu přesně říci, kdy nastane), spustil nějaký konkrétní úsek programu, který na tuto událost nějak má reagovat. Takových událostí bývá typicky v současných vestavěných systémech hodně. Jednotlivým událostem je třeba přiřadit adresy obslužných programů a také vyřešit situace, kdy se obsluhy dožaduje více událostí. Pokud rozumíte tomu, jak je to vyřešeno, tato otázka byla velmi triviální.

Největší potíže činilo stanovit adresu, na které začíná obsluha přerušení vyvolaného zadaným zdrojem. Ojedinělé potíže byly spojeny se stanovením pořadí provádění obsluh přerušení a pochopením vztahu mezi číslem IRQ a nulováním a od/maskováním přerušení v příslušných registrech).